HYPERGAME ANALYSIS OF CYBER SYSTEMS

THESIS

Kebin Umodu, Capt, USAF

AFIT-ENG-MS-17-M-075

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-075

HYPERGAME ANALYSIS OF CYBER SYSTEMS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Kebin Umodu, B.S.C.S.

Capt, USAF

March 2017

AFIT-ENG-MS-17-M-075

HYPERGAME ANALYSIS OF CYBER SYSTEMS

THESIS

Kebin Umodu, B.S.C.S.
Capt, USAF

Committee Membership:

Dr. G. Lamont
Chair

Dr. B. Borghetti
Member

Dr. B. Mullins
Member

AFIT-ENG-MS-17-M-075

Abstract

As cyber-physical systems (CPS) are increasingly being integrated into many facets of modern day life, attackers are gaining new vectors to launch their attacks on these networks. Network administrators need better tools to help support secure networks which can anticipate the attacks even when they are uncertain of an attacker's techniques. Hypergame theory is a method that models these cyber attack scenarios where players can have different perceptions about the game being played. The normal form hypergame is capable of comparing different strategies a player can select to improve their expected outcome based on their confidence in these perceptions. Capturing perceptions improves the accuracy of the model and incorporates the effects of a player's misconceptions and deception. Using this additional information a network administrator can formulate improved plans to better protect a CPS network. AFIT's Hypergame Analysis Tool (HAT) software can model these competitive scenarios in a normal form hypergame and capture multiple perceptions of a player. HAT can also scale the expected payoffs by the belief in being outguessed by the opponent, to further determine best strategies. This investigation analyzes HAT's ability to determine the best strategy for a network administrator defending a CPS. A new cyber defense modeling framework (CDMF) is used to create an easy to understand model for a cyber scenario. HAT is upgraded with the ability to calculate Nash equilibriums (NE) using a mixed integer programming method (MIP). MIP simplifies alternating NEs to analyze different strategies in HAT. HAT's validated MIP implementation is used to analyze cyber models and experiment on the Blotto game and CDMF model by changing NEs and row players. These experiments validate the hypothesis that analyzing hypergames in new and interesting ways results in new information and in-

sight that can support a CPS network administrator in making better defense plans against an unpredictable attacker.

Acknowledgements

I would like to thank my academic advisor Dr. Lamont for his tremendous persistent help with this paper. Thanks to Dr. Kovach for his continued assistance in this effort along with my committee members. I like to give a special thanks to Captain Ortiz for helping me stay focused. I like to thank my entire Umodu family for everything they done to help. Thanks to my dad for his encouragement throughout the process. Special thanks to my mom for constantly checking on me to make sure I was on task and making sure I had everything I needed. I also like to thank God for the ability to accomplish everything I have done and the amazing opportunity.

Kebin Umodu

Table of Contents

Page

List of Figures

List of Tables

Table Page

I. Introduction

This investigative effort expands on the ways hypergame analysis is conducted, particularly in the field of cyber-physical systems. The purpose of this research is to achieve better input for plan forging by a network defender. Hypergame analysis is an extension of game theory that allows a planner to incorporate ideas of misconception and deception into a model. This makes it more realistic and provides more representative information that can lead to more efficiency and greater payoffs when applied in the real world. This research explores how the AFIT Hypergame Analysis Tool (HAT) can be updated and utilized in new ways to find even more interesting information that can benefit a planner. The following sections introduce the main topics covered in this research, including cyber conflicts, game theory, the motivation behind the research, the main problem, the research goal, and the thesis structure.

1.1 Cyber Conflicts

In today's world the idea of cyber has permeated almost every aspect of our daily lives. This has allowed for unprecedented speed, efficiency and ease of services and communication in our modern world. Cyber technology has even found its ways into things where one might have previously asked "why?", such as cars, television sets, watches, eye glasses, and even refrigerators. As the tendency to add the "smart" prefix to more and more objects continues, steadily bringing everything into the "Internet of things", it becomes increasingly important to reconsider the security of connected networks from attackers. This is because as more devices are connected to the Inter-

net the attack surface increases as well. Although cyber security experts have been fighting to keep these networks protected via traditional and new methods, society continually sees instances where these systems are being circumvented and people are still falling victim to cyber attacks. One recent example of this was during the 2016 U.S presidential election where it was reported that Russian intelligence operatives had infiltrated emails of a candidate's party member [1]. The emails released to the public as a result of this attack exposed potentially damaging details about the internal activities within the political party and are believed to have swayed public opinion in the election [2]. Incidents such as this are simply a reminder that the cyber world is not as secure as society would like it to be and continuous efforts are needed to combat cyber attacks. One way to improve cyber security is to better understand cyber attacks. Cyber attacks can be seen as conflicts between a hostile entity and the system they are trying to affect along with those that are trying to protect it. By analyzing these conflicts, network defenders can gain better insight into how a hostile entity acts against a system as well as how the system and its defender should react to maximize system operability and minimize negative impacts.

1.2 Game Theory

A good way to analyze this conflict is through the use of game theory which is a mathematical approach to modeling conflicts between players. The ideas of game theory are known to be first formally recorded when mathematician Charles Waldegrave wrote a letter to a colleague discussing the best strategy for a two player card game known as "le Her" [3]. Charles discovered that an optimal strategy for a player was not to stick to any single strategy but to bounce between different strategies with a certain probability in what would come to be known as a mixed strategy. After Charles Waldegrave's insight into game theory, many others continued

to utilize game theory concepts to explain economic and social issues as well as more traditional game strategies such as chess. In 1928 Jon Von Neumann published a paper that truly defined game theory as its own mathematical study and formalized a lot of the concepts [4]. Since then game theory has continued to expand introducing new ways to model various conflicts and interactions including those in the cyber domain. In 2017 The former chair of the US House Intelligence Committee stated that game theory was consistently used in his department when making decisions on cyberattack policies [5]. Game theory's ability to help determine how rational players would interact in a given scenario have proven to be beneficial in understanding what an attacker might do to a network. Once an attacker's strategy is better understood, a defender can effectively plan on how to go about mitigating these efforts.

1.3 Motivation

In order for a network defender to formalize a good plan on how to protect their assets, they first must have good information. This information can come from modeling a cyber scenario using game theory, but the information that is gained is limited to the quality of the model itself. Therefore, in game theory the closer the model is to reality while still being reasonable and understandable, the better the information that can be garnered from it. As stated earlier traditional game theory was restricted in what it could model as it focused on complete information games where each player knew exactly what the other players' strategies were. When it comes to engagements in cyberspace however, the defender may not know the full capabilities of the attacker and vice versa. This means, using a model that inaccurately captures the situation can diminish the usefulness of the information gained and render that information limited in application. Even worse, this information can lead to players making decisions that result in sub-optimal outcomes. Many researchers expanded game theory

to address these short comings. This led to the development of hypergame theory which allows games of incomplete information to be modeled. In particular hypergame models are able to capture scenarios where players are not always fully aware of each other's available strategies. This leads to players possibly having a different outlook on the game, which in turn, influences there chosen strategies. With hypergame theory, cyber conflicts where a defender may not know all the attacker's strategies and vice versa, could be more accurately modeled. This new model could then lead to better insight on how to properly plan for these situations in order to achieve the highest payoff. Another benefit of hypergame theory is that the information gained can help find strategies with better outcomes than ones found only using perfect information game theory concepts.

1.4 Research Problem

Over the years, there has been a limited amount of research into hypergame theory. It has been applied to many fields including military engagements [6], social issues [7], and e-commerce [8] to name a few. More hypergame research has also been done in the field of cyber [9][10][11] as of late which is promising to network defenders seeking to gain an edge on their networks. However this area of research still pales in comparison to the amount of cyber research done in more traditional game theory. This can attributed to hypergame theory being relatively new. Another difficulty that this kind of research is facing is the lack of well-established hypergame tools. Normal game theory has well documented tools such as Gambit [12], which allows games to be modeled in normal or extensive form and then solved, and a tool called GAMUT [13], which can create random games fitting certain criteria. These tools make it much easier to analyze game theory conflicts and draw conclusions. When it comes to hypergames however there are only a few set of tools that have been

developed, and they are not nearly as established. The tool HYPANT [14] was created to model hypergames according to the model proposed by Wang [15] where hypergames had different levels of perceptions. It was tested against classic games as well as games developed in hypergame research and shown to provide accurate but limited results. This software has also been utilized in a few other academic papers. Another hypergame tool is the AFIT Hypergame Analysis Tool (HAT) which is similar to Gambit as it models games in their normal form, but it can also model hypergames in normal form. This capability makes it a strong candidate to become an integral tool for hypergame analysis. HAT currently has basic functionality, but is still being developed to flesh it out.

1.5 Research Goal and Objectives

The goal of this research is to further develop the field of hypergame theory, specifically as it pertains to cyber systems by developing an improved modeling and analysis environment for hypergame models. In particular the focus is on new ways to analyze hypergames to gain more information for the decision maker and assist in planning for cyber defense.

This research explores how different factors affect the analysis of a hypergame model by looking at these models in new ways to glean new information that enhances a model's ability to support the decision maker. This research has a three main objectives. The first objective is to improve the HAT software functionality in order to better analyze hypergame models. Currently the HAT software can only find equilibriums using two methods, being Lemke-Howson [18] and support enumeration [23]. Adding another method allows different equilibrium solutions to be found thus enhancing the insight that the software can provide to a decision maker. The second objective is to present a new and interesting cyber warfare model. This model in-

corporates the idea of mission success and failure for the defender and the attacker. The third objective is to analyze this and other cyber models in new ways. This includes changing a game's equilibrium strategy as well as exploring the model from the perspective of both the defender and attacker. Although many models have been created for cyber, only one other case has used hypergame normal form (HNF) to look at the model from the attacker's perspective to gain insight for the defender [10]. Previous researchers using Vane's HNF have not yet considered how different hypergame strategies from are affected by changing the full game equilibrium strategy, which this research investigates [35]. The hypothesis is that looking at different equilibrium strategies in a hypergame and alternating the row player can uncover useful information for a defender of cyber networks that can be used in planning.

1.6 Research Approach

The HAT software structure is studied to find areas of improvement as well as understand how a new Nash equilibrium method can be implemented. Nash equilibrium methods are researched to see which one would make a good addition into HAT based on their unique properties and improvements over the existing methods. The software is validated against classical games and established models that deal with cyber defense. Existing game theory and hypergame theory models in the cyber field are studied, and a new cyber model is developed that provides a benefit over the current models. The new model is the analyzed using the improved HAT software as well. The improved HAT software is used to run experiments on cyber models to discover benefits from analyzing hypergames and interpreting the results in new ways.

1.7 Thesis Organization

This document begins with an introduction of the main topics to be discussed in Chapter I. Chapter II provides more background information into game theory, hypergame theory, cyber-physical systems and how they intersect. Chapter III discusses the specific methods and hypergame software improvements used to conduct the research, and introduces a new model for the defense of a cyber-physical system. Chapter IV validates the HAT software against classic games before being used to research different cyber defense models including the model created in the previous chapter. Chapter IV also contains the experiments and analysis of the new information can be learned from the data. Chapter V concludes the document with the final results of the experiments. It touches on contributions made to the field of hypergame theory in regards to cyber defense, and ends with suggestions for future work.

II. Background

Game theory allows for the analysis of games from a mathematical aspect which enables the explanation of practical and not so practical outcomes. This chapter discusses development of game theory, starting with decision theory and ending with hypergame theory. Hypergame theory builds upon both game and decision theory. Following this there is a section on cyber warfare which explores the realm of cyber-physical systems (CPS) and their vulnerabilities. This leads into a survey of the application of game theory and hypergame theory in the cyber defense arena, showcasing its benefits and potential.

2.1 Decision Theory

Extended Form.

To understand hypergame theory there must be an understanding of game theory which itself is best understood by initially discussing decision theory. Decision theory is based on the idea that given a scenario where a decision has to be made, a rational decision maker makes choices to achieve the best outcome. Stated another way decision theory looks into how a decision maker can rationalize over a set of possible actions to select the action or set of actions that would most likely give them the highest payoff in the end. In order to rationalize over these decisions effectively each possible outcome has to be given a value so different choices may be compared to each other. This value of the outcome to the decision maker is called the utility [16]. To better understand this, here is an example scenario where a kid has to decide whether they want to have pizza for lunch or a hamburger. The kid is the decision maker and must pick one of the two options which leads to one of the two outcomes. The best decision is based on how much each outcome benefits the kid. Say the out-

come with the kid having the hamburger would give the kid a utility of 10 while the outcome with pizza would give the kid a utility of 8. This shows that the kid likes both choices but is happier with a hamburger for lunch over pizza. Therefore, decision theory would rationalize that the kid would choose the hamburger to maximize their utility. These scenarios in decision theory are typically represented with a tree structure known as the extended form. At the root of the tree is the decision maker node with the different actions that can be taken from that point branching off of it. These actions take the decision maker to another state which also can have actions that branch off of them or none at all, making it a terminal state. A terminal state is a state where no more actions can be taken and are more commonly referred to as leaf nodes to fit the tree motif. These leaf nodes are the only states that give a final utility to the decision maker. Figure 1 shows the lunch scenario in its extended form.

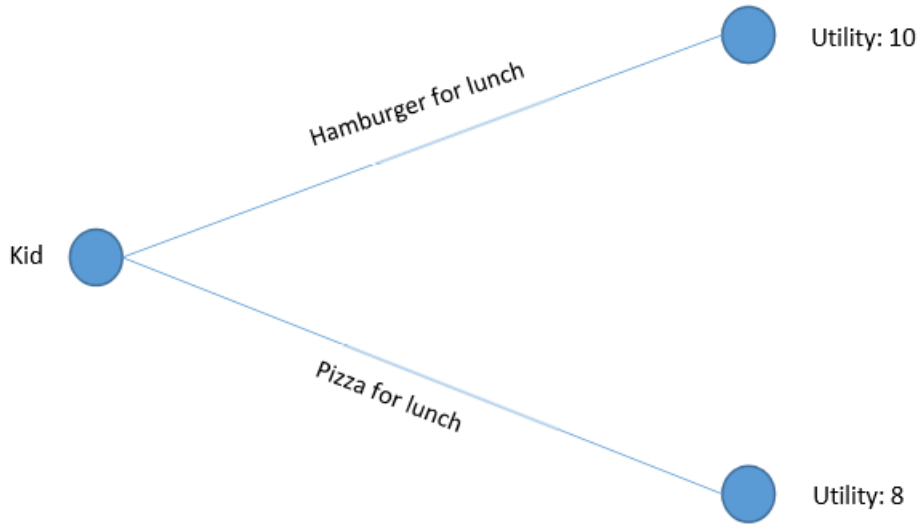

Figure 1. Decision Theory Extended Form.

Expected Utility.

When describing game theory earlier, it was stated that the decision maker chooses the action(s) that "most likely" get them the best utility. This was done purposefully

to express how decision theory can be used to make the most rational choice even when there is uncertainty. When there is uncertainty in a situation it means that the outcome of a particular action is not guaranteed. Instead there are multiple possible outcomes of the action, and each are given different probabilities with all of them summing up to 1. In this case the decision maker is trying to maximize their expected utility (EU) since they do not know for sure what utility they may receive. Consider the earlier scenario where the kid has to choose between having pizza or a hamburger for lunch, except this time the utility of their decision is affected on whether they are going to have pizza or hamburgers for dinner. In this scenario the kid would prefer to have a different type of food for each meal rather than the same thing for lunch and dinner. For this example there is a 70% chance that the kid has the same thing for dinner if he chooses to have pizza for lunch and a 40% chance the kid has the same thing for dinner if the kid chooses to have a hamburgers for lunch. This updated example is seen in Figure 2.

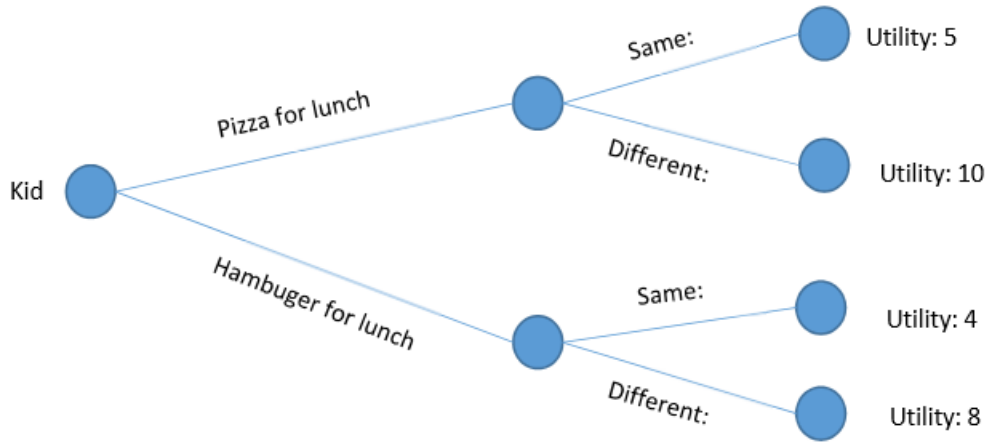

Figure 2. Decision Theory Extended Form with Stochasticity.

Now the choice of the decision maker is no longer as straightforward as picking the biggest number, but the rational action is still that which maximizes their EU. The

equation for EU is $EU = \Sigma_{i=1}^{n} p_i x_i$ where p_i is the probability of the i^{th} event and x_i is the utility of that event. To find the best decision for the kid's lunch, the EU for each decision is calculated by plugging the values into the equation. The EU for the pizza is $5 * 0.7 + 10 * 0.3 = 6.5$ and the EU for the hamburger is $4 * 0.35 + 8 * 0.65 = 6.6$. In this case the EU for eating a hamburger, 6.6, is slightly greater than the EU for eating a pizza for lunch, 6.5. Therefore, the kid should still choose to have a hamburger for lunch since the kid is less likely to have that same thing for dinner and therefore receive a higher EU. This example demonstrates how decision theory is able to help decision makers determine the best choice even in the absence of certainty. Now that the basics of decision theory have been explained, it is time to describe game theory.

2.2 Game Theory

Extended Form.

Similar to decision theory, game theory helps in making decisions. The big difference is game theory deals with two or more decision makers and the interactions of their strategies to determine the utilities achieved by all players. Scenarios in game theory are known as games. A game is formally defined as having a finite set of N players. Each player has a set, S_i, of pure strategies $\{S_1, S_2, ..., S_n\}$. Each player also has a set, v_i of payoff functions $\{v_1, v_2, ..., v_n\}$. Each payoff function assigns a value to each combination of selected strategies. Player i's payoff function is $v_i : S_1 \times S_2 \times ... \times S_n \rightarrow \mathbf{R}$ [16]. This means player's actions combine to produce different values for the player. The Colonel Blotto game introduced by Émile Borel in 1921 [17] is an example of a classic game in game theory. In this game two colonels each have an equal finite number of troops and have to decide how to allocate their troops across a set finite number of fronts. The colonel who has more units on a front wins that front and which ever colonel wins the most fronts wins the entire game. Al-

though the concept of the game is simple, the game provides interesting insights into how players can mix strategies or even gain advantages over their opponents which is an idea that is touched on later in Chapter IV. This section discusses the different aspects of two-player games. Although game theory can be applied for 2 or more players, traditionally 2-player games are analyzed. This reduces the complexity and simplifies the analysis of the different strategies of the game. Games in game theory are typically modeled in one of two ways. The form that is similar to the way decision theory is modeled is the extensive or tree form. This form sees one player being at the root node with their available actions branching off and the terminal nodes of the tree holding the payoffs. The difference is that at each layer of the tree, the player who is making the actions switches and the utilities displayed at the terminal nodes are for all players involved. A simple two player game example of this format can be seen in Figure 3. Player 1 is at the root of the tree and has actions available branching off. These actions lead to one of Player 2's decision nodes which have their available actions branching off of them or they themselves are terminal nodes. At the terminal nodes there are a pair of values. The first value is the utility for Player 1, and the second value is the utility for Player 2.

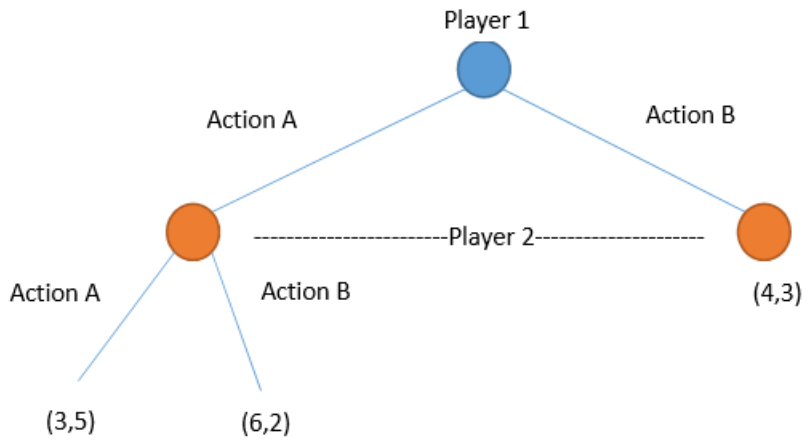

Figure 3. Game Theory Extended Form.

Figure 3 displays that if Player 1 wants a chance to maximize their utility with a payoff of 6, they have to take action A. However, this is not guaranteed as Player 2 may select action A, which gives Player 1 a payoff of 3 instead. This payoff of 3 for Player 1 is less than the guaranteed payoff of 4 if Player 1 selects Action B instead. This demonstrates the strategies associated with game theory where a player must take into account the decisions of other players when making their choice of action. This example also relates back to John von Neumann's proof of the min-max strategy where a player makes decisions that minimize their maximum loss potential.

Normal Form.

Another way games can be modeled is through the normal form. In this form an $m \times n$ matrix is formed. The m is the number actions available to Player 1 and n is the number of actions available to Player 2. Each cell holds the utility values for the actions corresponding with its row and column being played against each other. Table 1 shows the game from Figure 3 transformed from extensive form into the normal form.

The normal form allows the utilities for the different actions taken by the players to be seen more easily. For example, if Player 1 took Action A and Player 2 took Action B, Player 1 would receive a utility of 6 and Player 2 would receive a utility of 2. Although the extended form of the game can be translated into a normal form game, the extended form game is preferred when modeling games where players take turns. The normal form game is preferred when modeling games where each players actions are simultaneous.

Table 1. Game Theory Normal Form

Player 1 \Player 2	Action A	Action B
Action A	(3,5)	(6,2)
Action B	(4,3)	(4,2)

Pure Strategy Nash Equilibrium.

One of the main objectives in game theory is to find the actions that a player should take to maximize their EU with regards to other players trying to do the same. These actions place the game in a state of equilibrium and finding them is known as solving the game. A state of equilibrium is achieved when all players involved have no incentive to deviate from the actions they have selected. This state is referred to as the Nash Equilibrium. Steven Tadelis defines the Nash equilibrium as a profile of strategies for which each player is choosing a best response to the strategies of all other players [16]. The term Nash equilibrium is named after John Nash who discovered in his research that every finite normal form game has at least one equilibrium state or Nash equilibrium with some games having more. Considering the example normal form game in Table 1 the Nash equilibrium can be found by finding the actions that each player can take where they have no incentive to deviate from that action given that the other player maintains their strategy. Looking at the normal form game in Table 1, it shows a Nash equilibrium is found when Player 1 selects Action B and Player 2 selects Action A. The payoff for this strategy is 4 for Player 1 and 3 for Player 2. If Player 2 maintains Action A and Player 1 changes their strategy to Action A, Player 1 achieves a new payoff of 3 which is less than what they had before. Similarly, if Player 1 maintains Action B and Player 2 changes their strategy to Action B, Player 2 achieve a new payoff of 2 which is less than what they had before. This demonstrates that neither player has an incentive of changing their strategy to get a better payoff.

Mixed Strategy Nash Equilibrium.

In game theory there exist two types of Nash equilibrium strategies. The Nash equilibriums found previously are known as pure strategy Nash equilibriums. A pure

strategy Nash equilibrium is when an equilibrium is achieved after each player takes one particular action. However, there exists games where no matter what action is selected by a particular player there is always an incentive for the other player to change their strategy. An example of this can be seen in the classic Rock, Paper, Scissors game in Table 2.

Whichever action each player chooses initially, at least one of the players has an incentive to change their action given the other player keeps their chosen strategy. Although it is true that this game does not have a pure strategy Nash equilibrium it has what is known as a mixed strategy equilibrium. In a mixed strategy [16] Nash equilibrium each players strategy consists of randomly choosing their available actions with a certain probability. Equilibrium is achieved because players must randomize their actions at a certain probability to account for the uncertainty of the other players actions. When solving a mixed strategy equilibrium, the specific probability of a players actions, over which they randomize, are found. In order to solve a mixed strategy Nash equilibrium, it must be known which actions actually have a chance of being selected as part of the mixed strategy. That means for any particular action it must have a probability greater than 0 of being randomly selected. These actions that meet this criterion are known as belonging to the support [16] of the particular mixed strategy. Knowing the support in a mixed strategy allows the game to be solved. Unfortunately the support is not always easily known simply by looking at the game itself. When the support for a mixed strategy cannot be easily seen, as it is for larger and more complex games, the mixed strategy must be calculated using mathematical

Table 2. Rock, Paper, Scissors Game

Player 1 \Player 2	Rock	Paper	Scissors
Rock	(0, 0)	(-1,1)	(1, -1)
Paper	(1, -1)	(0,0)	(-1,1)
Scissors	(-1,1)	(1, -1)	(0,0)

techniques such as linear programming. Rock, Paper Scissors is a relatively simple game and using a little intuition about the game and its payoffs it can be assumed all the actions for each player are included in the support. To finding the mixed strategy for a game means finding the probabilities for the actions where the other player is indifferent to their choice of actions. Therefore given the probability of Player 2s mixed strategy, Player 1 is indifferent on their choice of mixed strategy.

This can be modeled by first representing the distribution over the actions as variables p_1 for Rock, p_2 for Paper, p_3 for Rock. Since the probabilities must sum to 1 it can be said that $p_3 = 1 - p_1 - p_2$, as is seen in Table 3. In order for Player 1 to be indifferent over their actions they have to expect the same utility no matter which action they choose assuming Player 2 is using a particular probability distribution over their actions. To show this, set the EUs for every action for Player 1 equal to each other. EU is calculated by the equation $EU_j = \Sigma_{i=1}^{n} probability_i * utility_{ij}$ where i is the opponent action and j is the action for which the EU is being calculated. The EUs for Player 1 actions are $EU_{Rock} = (0) * (p_1) + (-1) * (p_2) + (1)(1 - p_1 - p_2)$, $EU_{Paper} = (0) * (p_1) + (-1) * (p_2) + (1)(1 - p_1 - p_2)$, and $EU_{Scissors} = (0) * (p_1) + (-1) * (p_2) + (1)(1 - p_1 - p_2)$. As stated earlier each player is indifferent to their actions because they can expect the same utility. Mathematically this $EU_{Rock} = EU_{Paper} = EU_{Scissors}$. The probabilities can be solved by plugging in the equations for the EUs and solving for individual variables. The result of this is $p_1 = p_2 = p_3 = 1/3$ which is the probability distribution for Player 2's mixed strategy. The same method is

Table 3. Solving Rock, Paper, Scissors Mixed Strategy

	p_1	p_2	$p_3 = (1 - p_1) - p_2$
Player 1 \Player 2	Rock	Paper	Scissors
Rock	(0, 0)	(-1,1)	(1, -1)
Paper	(1, -1)	(0,0)	(-1,1)
Scissors	(-1,1)	(1, -1)	(0,0)

used to find the probability distribution for Player 1's mixed strategy which turns out to be 1/3 for all actions as well. This fits the natural intuition of the game where no action has an advantage over the other when one player does not know what the other player is doing. It is important to remember that this method worked due to our assumption of every action being in the support turned out to be correct. This is not the case for all games and for those games they can be solved more efficiently with some of the more complex algorithms that have been developed over the years such as the Lemke-Howson method [16].

Nash Equilibrium Algorithms.

The example in the previous section showed how to find the mixed strategy of a two-player game. Unfortunately this method does not always work with more complex games. This is because in order for this method to work the support of the mixed strategies, which are the actions in a player's mixed strategy which have a probability greater than 0%, had to be assumed. In order to find the mixed strategy Nash equilibrium for a game where the support is unknown different algorithms must be used. Each of these algorithms that have been developed have their unique properties and strengths.

Lemke-Howson Method.

One of the earliest and most popular algorithms used to find the mixed strategy is the Lemke-Howson algorithm developed by C. E. Lemke and J. T. Howson Jr. in 1964 [18]. Their algorithm was capable of finding a mixed strategy Nash equilibrium in two player non-degenerate games using linear programming techniques. Although the algorithm itself is not very complex, the amount of time it takes to find an equilibrium grows exponentially with the size of the problem. Although this is one

of the algorithm's weaknesses one if its greatest strengths is its guarantee to find an answer if properly implemented. The following is an example of how the mixed strategy for a game could be solved using the Lemke-Howson algorithm [19]. For this example, consider the two player $m \times n$ game in Table 32.

The algorithm starts by generating two matrices for each player which are matrix A and B corresponding to player A and B's utilities respectively.

$$A = \begin{vmatrix} 2 & 4 & 0 \\ 0 & 0 & 3 \\ 3 & 2 & 2 \end{vmatrix} \qquad B = \begin{vmatrix} 3 & 2 & 0 \\ 2 & 4 & 2 \\ 0 & 0 & 4 \end{vmatrix}$$

The mixed strategies found by this algorithm are in the form of a m sized vector, x, for player A and a n sized vector, y, for player B. According to the algorithm the constraints for a Nash equilibrium solution are $A^i y \leq 1$ and $x^T B_j \leq 1$. Non-negative slack variables r and s are used to turn these inequalities into the equalities $Ay + r = 1$ and $B^T x + s = 1$. Two tableaux are set up that solve for the slack variables as seen in tableau A and tableau B.

$$r_1 = 1 - 2y_4 - 4y_5 \tag{A1}$$

$$r_2 = 1 - 3y_6 \tag{A2}$$

$$r_3 = 1 - 3y_4 - 2y_5 - 2y_6 \tag{A3}$$

Table 4. Lemke-Howson Example Game

Player A \Player B	4	5	6
1	(2, 3)	(4,2)	(0, 0)
2	(0, 2)	(0,4)	(3,2)
3	(3,0)	(2, 0)	(2,4)

$$s_4 = 1 - 3x_1 - 2x_2 \tag{B1}$$

$$s_5 = 1 - 2x_1 - 4x_2 - 2x_3 \tag{B2}$$

$$s_6 = 1 - 4x_3 \tag{B3}$$

In this algorithm x and r are seen as pairs along with y and s, and labels refer to the index of a variable solved for in an equality. For example the equality $r_1 = 1-2y_4-4y_5$ provides the label 1 from the index of r. The goal of the algorithm is to start with all the labels and then pivot the variables until all the labels are covered again by the union of the two tableaux. For this example the labels are 1 through 6 and as stated earlier the algorithm starts with all the labels. A pivot is performed by choosing a random index to solve for which causes a label to be lost. The lost label becomes the new index to solve for and this is repeated until all the labels are accounted for. For example If the previous equality is solved for y_4 it becomes $y_4 = \frac{1}{2} - \frac{r_1}{2} - 2y_5$. In this equality, the label is now 4 and the label 1 is lost. When choosing which equality to solve an index for the general rule is to choose the equality with a minimum value for that index. If there is a tie than randomly pick one. To avoid unending loops, if the tie is formed again break the tie differently. The first index to pivot can be random, and this example solves for x_1 in tableau B. The minimum value rule means use equation $B1$. This creates the new tableau B‘.

$$x_1 = \frac{1}{3} - \frac{2}{3}x_2 - \frac{1}{3}s_4 \tag{B‘1}$$

$$s_5 = \frac{1}{3} - \frac{8}{3}x_2 - 2x_3 + \frac{2}{3}s_4 \tag{B‘2}$$

$$s_6 = 1 - 4x_3 \tag{B‘3}$$

After pivoting on x_1 and plugging in the variable for the other equations, label 4 is lost since there is no longer an equation solving for a variable with an index of 4 which used to be s_4. In order to get it back, pivot on y_4 in tableau A because y and s are pairs. The minimum value rule dictates that the $A3$ be used. This creates the new tableau A‘.

$$r_1 = \frac{1}{3} - \frac{10}{3}y_5 + \frac{2}{3}y_6 + \frac{1}{3}r_3 \tag{A‘1}$$

$$r_2 = 1 - 3y_6 \tag{A‘2}$$

$$y_4 = \frac{1}{3} - \frac{2}{3}y_5 - \frac{2}{3}y_6 - \frac{1}{3}r_3 \tag{A‘3}$$

This process is repeated until all labels are accounted for with the union of both tableau as seen in Table B“‘1 and A“‘1. See Appendix A for the full step-by-step calculations.

$$x_1 = \frac{7}{24} - \frac{5}{12}s_4 + \frac{1}{8}s_6 + \frac{1}{4}s_5 \tag{B“‘1}$$

$$x_2 = \frac{1}{16} + \frac{1}{8}s_4 - \frac{3}{16}s_6 - \frac{3}{8}s_5 \tag{B“‘2}$$

$$x_3 = \frac{1}{4} - 1 - 4x_3 \tag{B“‘3}$$

$$y_5 = \frac{1}{6} - \frac{1}{15}r_2 + \frac{1}{10}r_3 - \frac{3}{10}r_1 \tag{A“‘1}$$

$$y_6 = \frac{1}{3} - \frac{1}{3}r_2 \tag{A“‘2}$$

$$y_4 = \frac{12}{45}r_2 - \frac{6}{15}r_3 + \frac{1}{5}r_1 \tag{A“‘3}$$

Now the mixed strategy equilibrium probabilities can be found by solving for the variables in the equation. This is done by assigning a 0 to all the slack variables. The result of this can be seen in equation 9.

$$\begin{aligned} x_1 &= \tfrac{7}{24} \\ x_2 &= \tfrac{1}{16} \\ x_3 &= \tfrac{1}{4} \\ y_5 &= \tfrac{1}{6} \\ y_6 &= \tfrac{1}{3} \\ y_4 &= 0 \end{aligned} \tag{1}$$

These values need to be normalized by dividing each value by the sum of its entire vector. The final values are $x_1 = 0.482$, $x_2 = 0.103$, $x_3 = 0.413$ for the x vector and $y_4 = 0$, $y_5 = 0.333$, $y_6 = 0.667$ for the y vector. These results give an equilibrium mixed strategy for the game. Given the right set up this algorithm is guaranteed to work on any non-degenerate two player game. If the game is degenerate, ties can form during the algorithm between equalities with the same minimum index value. These problems can still be solved using small perturbations to break these ties which can be done in polynomial time. If the game involves more than one player than other methods have to be used to find the mixed strategy. Also this method essentially only finds one Nash equilibrium per run. In order to find all the mixed strategies every possible pivot combination would need to be tested which can result in a lot of computation time in large problems. Other methods may be able to find all the Nash equilibriums more efficiently.

Other Methods.

In 1987 two of researchers introduced the Simplical Subdivision method as another way to find mixed strategy Nash equilibriums [20]. This method works by solving the nonlinear complementarity problem (NLCP) defined over the product of unit simplexes that form when calculating the Nash equilibrium in a non-cooperative game. Similar to the idea of labeling found in the Lemke-Howson method, the algorithm finds a completely labeled simplex in a simplical subdivision of a single properly labeled unit simplex. The algorithm is setup by considering an N player game where $n_j + 1$ is the number of pure strategies available to player $j \in I^N$. The product of the index sets I^{n_j+1} is represented by K and a vector $k = (k_1, ..., k_n) \in K$ represents the pure strategy vector where player j plays pure strategy $k_j \in I^{n_j+1}$. The loss player j experiences when playing k is captured in the variable $a^j(k)$ where $\forall k \forall j$, $k \in K$, $j \in I^N$, $a^j(k) > 0$. The mixed strategy space for player j is represented by S^{n_j} and the simplical product $S = S^{n_1} \times ... \times S^{n_N}$ is the strategy space for the non-cooperative game. The value of x_{jh} where $x \in S$ represents the probability player j plays the pure strategy $h \in I^{n_j+1}$. The expected loss to player j if x is played is stored in p_j where $p^j(x) = \sum_{k \in K} a^j(k) \prod_N^{i=1} x_{ik_i}$ The expected loss to player j if strategy h is played instead of x_j when other the players' strategies stay the same is calculated by $m_h^j(x) = \sum_{k \in K_{jh}} a^j(k) \prod_{i=1, i \neq j}^N x_{ik_i}$, where $K_{jh} = \{k \in K | k_j = h\}$. An equilibrium strategy $x \in S$ is obtained when $\forall j \forall h$, $j \in I^N$,$h \in I^{n_j+1}$ $p^j(x) \leqslant m_h^j(x)$. This formulation highlights a unique aspect of this algorithm which is its focus on the potential loss a player would experience if they played a certain mixed strategy against their opponent's strategy. One of the other benefits of this method is that it can be used to solve a game where there are more than 2 players.

Enumeration of All Extreme Equilibria.

Another method used to find the Nash equilibrium in bi-matrix games is through the enumeration of all extreme equilibria (EEE) [21], and like most Nash equilibrium finding methods, it makes use of linear programming to calculate the optimal mixed strategy for each player. Extreme equilibria are the vertices of the polytopes that are formed from the best responses of one player to the given strategy of another player. This algorithm enumerates all the vertices and checks to see if the mixed strategies found at these extreme equilibria meet the optimality conditions to be considered a Nash equilibrium. The first optimality condition that must be satisfied to consider a pair of best response strategies $(\hat{x}, \hat{y})$ a Nash equilibrium is $\hat{x}^t A\hat{y} = \hat{\alpha}$ and $\hat{x}^t B\hat{y} = \hat{\beta}$. This condition states that the values for α and β are equal to the payoffs for Player 1 and Player 2 respectively when they play their best response strategies. The second condition that must be met is $\hat{x}^t(1\hat{\alpha} - A\hat{y}) = 0$ and $(\hat{\beta}1^t - \hat{x}B)\hat{y} = 0$. This condition states that the equilibrium is found only when a strategy is played with 0 probability or it is the best response to the other player's strategy. One of the benefits of this algorithm is that it is able to find all the Nash equilibriums in a game. One of the drawbacks is that it formulated only for two player games. The algorithm was improved upon in [22] by using integer pivoting as opposed to floating point numbers to increase precision and by improving the check for degenerate games to make the algorithm run faster.

Enumeration of the Support.

Continuing on with the idea of enumeration, one of the simplest methods for finding the Nash equilibrium is through the enumeration of the support [23]. This algorithm enumerates all the possible supports for two players given that the support sizes for each player's strategy are the same size. For example if Player 1 had 3 pure

strategies and Player 2 had 2 pure strategies the algorithm would search for Nash equilibriums where both players have a support size of 1 and where both players have a support size of 2. Looking for the equilibrium where both players have a support size of 1 would involve the algorithm considering 6 different combinations of actions to see if any of these combinations are an equilibrium. After all these combinations are checked for an equilibrium, the algorithm searches strategies where both players have a support size of 2. Since the support sizes have to be the same the algorithm only goes up the size of the player with the least amount of actions. Given a support size and a combination of actions in those supports, linear equations are solved using simple LU decomposition and back substitution to determine the values for the mixed strategy. Once the mixed strategies are determined they are checked to see if all the values are positive as their cannot be any negative probability values. If the mixed strategy passes that test than it is checked to see if the payoffs are maximized and equal for the strategies in the support which is found via the equation Ay for Player 1 and x^tB for Player 2. This algorithm is really easy to implement, but is limited to only two player non-degenerate games. Another issue with the algorithm is that it cannot find equilibrium strategies with unequal supports.

Global Newton Method.

The Global Newton Method (GNM) is another algorithm for finding the Nash equilibrium in a normal form game [24]. Along with the linear algebra found in other algorithms, this algorithm utilizes concepts of homotopy and vector calculus. The algorithm works by using a game instance to develop a system of equations whose zeros are to be computed. This system of equations is then deformed into another system of equations that has a unique and much more easily computed solution. This deformation is then reversed and unique solutions are found tracing back along the

path to the original system of equations defining the original game. These solutions are in turn interpreted into the equilibrium mixed strategies for the players. The benefit of this method is that it can be used to find multiple equilibriums in one run using the input vector that defines the ray for the path of the homotopy. The algorithm is also capable of handling n-player games. One of the drawbacks however is that it does not distinguish degenerate games so it is not guaranteed to find all equilibriums in those instances.

Iterated Polymatrix Approximation.

The next method explored is known as iterated polymatrix approximation (IPA) was developed in concert with the GNM as a way to improve its effectiveness although it has been shown that IPA can stand on its own as well [25]. IPA works by first constructing a polymatrix game based off the payoffs players receive for playing certain strategies against others. The algorithm is based off the property that a mixed strategy is an equilibrium of an original game if and only if it is also an equilibrium of the polymatrix game that approximates it. This algorithm also makes use of a variant of the Lemke-Howson method to calculate an equilibrium. The main purpose behind the development of this algorithm was to quickly find a equilibrium to provide a starting point to search for further equilibriums using the GNM. This is because although it is faster than the GNM, it can only find one Nash equilibrium per run and it also theoretically less reliable with a higher likelihood to stall as compared to the GNM. Despite these setbacks it is still offered as a way to calculate Nash equilibriums in software such as Gambit.

Lyapunov Function.

A unique method to find Nash equilibrium in n-player games was developed using a Lyapunov function [26]. This method finds a single Nash equilibrium by defining the game in terms of a Lyapunov function and then, similar to other linear programs, minimizing the objective function subject to constraints on the probabilities of the strategies. Research has shown that the algorithm is slower compared to other algorithms such as the Lemke-Howson method. This is a result of the algorithm being vulnerable to finding local minimums in its search that do not actually result in a Nash equilibrium. Therefore the algorithm could take longer if its starting point is surrounded by local minimums, but this also means that the algorithm could perform potentially well if the starting point for the search is close to a Nash equilibrium. The major strength of this algorithm is that it is capable of finding Nash equilibriums for n-players.

Linear Complementarity Problem.

Sometimes Nash equilibriums must be found for games are in the extensive form. With the algorithms mentioned so far, in order to find the Nash equilibrium of this kind of game it would first have to be converted to a normal form game. The problem with this is that many times this conversion is costly and can cause the game to grow much larger. This is why methods to solve games in their current extensive form were developed. One such method utilizes the idea of linear complementarity problems (LCP) to find the Nash equilibrium efficiently [27]. This method works by employing Lemke's algorithm on solving LCPs in general [28]. First the extensive form game is interpreted into its sequence form. This is done by taking every leaf node in the game to define a sequence of choices that result in a particular payoff. From here the goal is to find the realization plan which represents the probabilities a player would select

a certain sequence as a best response to another player. Once the game is defined in sequence form Lemke's method is used to solve the LCP. This method is proven to be exponentially more efficient at computing extensive form games in their current state than first converting to normal form. This algorithm can also be used to solve normal form games as well. This method is also capable of handling degenerate cases using lexicographic degeneracy resolution similar to the Lemke-Howson method, but a drawback is that it is restricted to games of two players.

Tracing Logit.

Another method that uses the idea of homotopy to find the Nash equilibrium is the tracing logit method [29]. This method works by considering quantal response equilibriums (QRE) for n-player games. A QRE is an equilibrium where the player takes into account some form of error when computing their best response. This error is calculated based on a joint distribution. This causes a player to pick a strategy that may deviate from the Nash equilibrium, but may pay off in practice for repeated games. When that error is chosen independently from an extreme value distribution a logit equilibrium can be found. In order to find the Nash equilibrium a branch of the logit equilibrium correspondence is traced out. This method is useful for calculating n-player games efficiently and makes heavy use of linear algebra for its calculations.

PNS.

Many of the methods mentioned so far approach the problem from different mathematical standpoints that can be seen as fairly involved. The issue with this is that these mathematical algorithms although complete in their ability to find a solution can lead to exponential computation times when solving larger problems. This was a large motivation for the development of the simple search PNS algorithm named

for the its authors Ryan Porter, Eugene Nudelman and Yoav Shoham [30]. The PNS algorithm approaches the problem of finding a Nash Equilibrium by searching within the solution space guided by simple heuristics. First the algorithm orders possible solutions by the support sizes by smallest to biggest. Then the each support is checked to see if a dominant strategy exists given the other players' actions. If this is the case then the solution cannot possibly be a Nash Equilibrium and the next set of supports is checked. If the supports do not have dominated strategies than the solution is checked for feasibility. The feasibility check attempts to solve a linear equation to find the mixed strategy values and see if they are in fact a Nash equilibrium. This method proved to be much faster than Lemke-Howson at finding Nash equilibriums across many game instantiations. The authors also developed a n-player version of the PNS algorithm using recursive backtracking and iterated removal of strictly dominated strategies which demonstrated a significant computation time speed up when compared to GNM. Both versions of the algorithm were also able to find solutions to certain large problems where their counterparts were unable. Other important aspects of this algorithm is that it was developed to find a single sample equilibrium and due to the nature of its search, the algorithm finds solutions with smaller sized supports.

Mixed Integer Programming.

The last algorithm discussed is the mixed integer programming algorithm (MIP) [31]. This algorithm works on 2-player normal form games and similar to the heuristics in the PNS algorithm it searches for solutions with characteristics associated with a Nash equilibrium. MIP introduces a new characteristic known as regret of a pure strategy. The regret of a pure strategy is the difference between the utility a player would receive when playing an optimal strategy given the other player's mixed

strategy versus playing that pure strategy. MIP works on the principle that in a Nash equilibrium all strategies are either played with 0 probability or played with 0 regret. In this method each player $i \in 0, 1$ has strategies $s_i \in S_i$. The utility of a strategy is represented as $u_i(s_i, s_{1-i})$ where s_{1-i} is the other player's chosen strategy and u_i is the utility function that returns a value. The probability a certain strategy is played is represented as p_{s_i}. The MIP method uses a binary variable b_{s_i} that is set to 1 if a strategy s_i is played with the probability $p_{s_i} = 0$ and b_{s_i} is set to 0 if the strategy is indeed in the support of the strategy. However if $b_{s_i} = 0$ than strategy s_i must be played with a positive probability and 0 regret. For each player there is a variable u_i that represents the highest possible EU that a player can achieve given another player's mixed strategy. Meanwhile the variable u_{s_i} represents the EU when a player plays a specific strategy against the other player's mixed strategy. Knowing the u_i and the u_{s_i} allows the regret r_{s_i} to be calculated for a specific strategy. Also there is a constant U_i for both players that indicates the maximum difference between two utilities a player can achieve in a game. Mathematically this is represented as $U_i = \max_{s_i^h, s_i^l \in S_i, s_{1-i}^h, s_{1-i}^l \in S_{1-i}} u_i(s_i^h, s_{1-i}^h) - u_i(s_i^l, s_{1-i}^l)$. In order to find a Nash equilibrium the values for the variables p_{s_i}, u_i, u_{s_i},r_{s_i}, b_{s_i} must be found under the constraints seen in Equation 2.

$$
\begin{aligned}
(\forall i) \sum_{s_i \in S_i} \quad p_{s_i} &= 1 \\
(\forall i)(\forall s_i \in S_i) \quad u_{s_i} &= \sum_{s_{1-i} \in S_{1-i}} p_{s_{1-i}} u_i(s_i, s_{1-i}) \\
(\forall i)(\forall s_i \in S_i) \quad u_i &\geq u_{s_i} \\
(\forall i)(\forall s_i \in S_i) \quad r_{s_i} &= u_i - u_{s_i} \\
(\forall i)(\forall s_i \in S_i) \quad p_{s_i} &\leq 1 - b_{s_i} \\
(\forall i)(\forall s_i \in S_i) \quad r_{s_i} &\leq U_i b_{s_i}
\end{aligned}
\tag{2}
$$

If all the constraints are met than the mixed strategy is a Nash equilibrium solution. One of the unique aspects of this algorithm is that an objective function can be added to the linear program to find an optimal Nash equilibrium based on the objective. The authors of this method also presented three additional ways the idea of regret could be used with different objective functions and constraints in order to turn MIP into an anytime algorithm. This means, the algorithm could be halted at anytime to return a ε-equilibrium solution, which is a solution that is close to equilibrium. MIP was shown to be faster than the Lemke-Howson method but slower than the PNS method on certain instances. However its unique ability to find optimal Nash equilibriums makes it a noteworthy algorithm. Table 5 is a quick reference chart of all the methods discussed in this section.

Table 5. Nash Equilibrium Algorithms

Algorithm	#players	#equilibriums per run
Lemke-Howson	2	1
Simplical Subdivision	N	1
Extreme Equilibria Enumeration (EEE)	2	All
Enumeration of the Support	2	Multiple
Global Newton Method (GNM)	N	Multiple
Iterated Polymatrix Approximation (IPA)	N	1
Minimizing the Lyapunov Function	N	1(not guaranteed)
Solving Linear Complementarity Problem	2	1
Tracing Logit	N	Multiple
Simple Search Method(PNS)	N	1
Mixed Integer Programming (MIP)	2	Multiple

Complete Information Games.

Game theory is a very rich field and there are many topics that can be discussed in more depth. Previous sections touched on one of the more important topics which is finding the equilibrium in a game, but there are a few other facets of game theory that are also worth more attention. First is the topic of games that have complete information. In a complete information game all the information about the game is known. This means, all the players and their possible actions are known along with all the possible results of the game and what actions guarantee those results. Also, the utilities each player receives from those results are known as well. Games are also broken down into perfect information games and imperfect information games. A perfect information game means that at any point in the game a player knows where they are in the game prior to their own action, meaning they know the moves that have been made before them and the result of that move, when the game is viewed in its extensive form. In an imperfect game the player may not know the action that the other player has chosen, or what the result of that action was, and therefore does not know what the result of their action is [32]. The game Rock, Paper, Scissors is an example of an imperfect complete information game, because due to the simultaneous actions taken by each player, neither player really knows what the results of their chosen action are going to be. This is more easily seen when the game is moved to its extensive form where the last player to move is unaware of what the other player has done by definition of the game. Games that are sequential in nature and thus normally represented by the extensive form have a chance to be perfect complete information games.

Dominant Strategies.

Another topic of game theory to discuss is the idea of dominant strategies. A dominant strategy is a pure strategy where no matter what the strategy of the other players, the dominant strategy is the best response. To help illustrate this point, look at one of the most popular games in game theory known as the Prisoners Dilemma. The scenario for this game is two partners in crime have to decide whether to tell on the other for a reduced sentence while subjecting their partner to more time or choose not tell on their partner and do more time than the reduced sentence. Table 6 shows the game modeled in normal form.

Table 6. Prisoner's Dilemma

Prisoner 1 \Prisoner 2	Dont Tell	Tell
Dont Tell	(-3, -3)	(-6, -2)
Tell	(-2, -6)	(-5, -5)

The table shows that if the prisoners work together they can each only get 3 years in prison which is only 1 more than if they had told on their partner. Therefore, it is beneficial to both prisoners if they cooperated and both did not tell on each other. The problem is that both players have dominant strategies that stop them from working together. From Player 1s point of view the strategy of telling dominates the other choice because no matter what Player 2 chooses, Player 1 always get a higher pay off by telling. The same goes for Player 2s dominant strategy. Unfortunately, by each player rationally playing their dominant strategy, the result is both Player 1 and Player 2 choose to tell and get 5 years in prison when they could have only gotten 3 years if they worked together and did not tell. The dilemma of both prisoners telling on each other demonstrates a dominant strategy equilibrium.

Zero Sum Games.

Another key concept of game theory is the zero sum game. In a two-player zero sum game every combination of actions from players involved has utilities that sum to zero. Rock, Paper Scissors is an example of a zero sum game because games result in either one player winning and the other losing or the game results in a tie. These games have special properties, which allow them to be solved more efficiently than other games. Also due to the game being zero sum sometimes when modeling these games the utility is displayed with only one number. This number usually represents the utility of the first player/row player. The other players utility is simply the negation of that.

Functional Utilities.

The last game theory topic to touch on is the idea of functional utilities. Some games have actual functions for a utility instead of a value. For these games the equilibrium may change as the evaluation of the function changes with the value of the variables used in those functions. To solve the game, the inflection points need to be found which are the values of the variable at which the equilibrium may change [33]. Figure 7 show the Hawk Dove game where two birds are fighting over food. The utilities represent the payoff expected when certain birds fight against each other. The variable V represents the value of the food and variable C represents the cost of the food. If the cost of the food is less than the food then the game contains two pure strategies and a mixed strategy equilibrium state. However if the cost of the food exceeds the value than the game turns into a variation of the Prisoner's Dilemma game, and the birds end up fighting over the food and achieve a suboptimal outcome.

Table 7. Hawk Dove Game

	Hawk	Dove
Hawk	$\frac{(V-C)}{2}, \frac{(V-C)}{2}$	V,0
Dove	0,V	V/2,V/2

2.3 Hypergame Theory

History of Hypergame Theory.

Over the years more research has gone into game theory, applying it to real world scenarios to understand the interactions between the players in the game and the outcomes. However, there are certain scenarios that game theory has had trouble modeling which are games of incomplete information. To address this inability researcher John Harsanyi developed a model for an incomplete information game [16] . This model was able to analyze games where a player may be unaware of the type of opponents they are facing and their type affects the payoffs. These kind of games became formally known as Bayesian games. Although these kind of games are able to explore scenarios that game theory could not support prior, there were still common real world games that Bayesian games did not address. This is why researcher Peter Bennet went on to establish the concept of hypergame theory [34]. Hypergame theory aims to model games where players may have different perceptions of the game and the other player's may or may not be aware of these perceptions. This allows for each player to have a different idea of what game they are playing. With hypergame theory, it is now possible to think about games where players are attempting to deceive their opponents about what their possible strategies are. The benefits of this model can be seen in the classic real world example of the Fall of France [6]. In this example France and Germany are fighting during WWII and Germany was on the offensive. France believed that Germany's options were to attack along the Maginot Line, which was heavily fortified, or attack from the north. Believing that Germany would not

attempt to attack the fortified Maginot Line, France placed emphasis on defending the north. France also positioned troops to respond to an attack on the Maginot Line if necessary. From France's perspective these seemed like the only feasible attack options for Germany. Germany initially planned to attack the north and use superior fighting to win the battle, but there was fear that this would result in a stalemate or a very costly victory. After some consideration Germany decided to focus on an attack plan through the Ardennes Mountains where France would not expect it. France disregarded this option because the mountainous terrain was considered too harsh for German tanks, but German officers determined their tanks were up to the task. It is also important to note that France got a hold of Germany's old plans thus strengthening their confidence in their planned strategy. Germany followed through with their new attack plan and claimed a decisive victory over the ill-prepared France. Germany set up a deceptive advance using the options that France expected and then launched a surprise attack from the option that France did not expect. The gamble paid off for Germany. Although Germany did not explicitly use a hypergame approach to model the battle and pick the best strategy, this shows how useful a hypergame model can be in modeling these situations where normal game theory falls short.

After Bennet introduced the hypergame theory there have been a few proposed models on how to capture the idea formally. The two most prominent models are the levels of perception model and then the hypergame normal form (HNF) model. In the levels of perception model there are different levels to the way a game is seen from each players point of view which is represented by H_i^L. L is the level of the hypergame and i is the player that the perception belongs to. The first level, H_i^0, represents the original perception of the game being played just as in normal game theory model. At this level perception of the game is not taken into account. The next level is H_i^1, representing each player's current view of the game. At this point each player

is possibly playing a different game although each player is not aware of the other players perception. The next level is H_i^2, and it is used to represent what a player believes another players view of the game is. This is where misconception comes into play because a certain player can be led to perceive something about another player that is untrue. Beyond this is H_i^3, which represents what a player believes another player is thinking about their perception of the game which introduces even more possibilities of misperception. These levels of perception could continue on but the author acknowledges that real world scenarios usually only get to this point so looking at further levels would not provide anything beyond mathematical significance [15]. This model allows the concept of incomplete information to be captured as a player can perceive a game of different options, strategies, preferences, higher order expectations and even players. However, the visualization of this model was a slight departure from the well-established extensive and normal form game models. This led to continued research into the idea of capturing misperceptions and deception in games of incomplete information, and brought on the development of the HNF model by Russel Vane [35]. Russel Vane understood that the real world is more likely to contain games with incomplete information and that basic game theory could not account for. He sought to create a game model that would allow a planner to anticipate being unsure of the other player's strategies and assess the risk associated with making decisions based off their belief of the opponent.

HyperGame Normal Form.

The HNF created by Vane was developed only for two players although Vane states that it can be expanded for more players. Table 8 holds a template for the HNF.

Table 8. Hypergame Normal Form

				C_Σ	S_1	S_2	S_3	...	S_n
$P_{(K-1)}$				$C_{(K-1)}$	c_{k1}	c_{k2}	c_{k3}	...	c_{kn}
	...			...	...	...	...	...	...
		P_1		C_1	c_{11}	c_{12}	c_{13}	...	c_{1n}
			P_0	C_0	c_{01}	c_{02}	c_{03}	...	c_{0n}
$R_{(K-1)}$	...	R_1	R_0=full game	row\col	col 1	col 2	col 3	...	col n
r_{k1}	...	r_{11}	r_{01}	row 1	u_{11}	u_{12}	u_{13}	...	u_{1n}
r_{k1}	...	r_{12}	r_{02}	row 2	u_{21}	u_{22}	u_{23}	...	u_{2n}
...	...	...	...	...	...	...	...	...	...
r_{km}	...	r_{1m}	r_{0m}	row m	u_{m1}	u_{m2}	u_{m3}	...	u_{mn}

The model has been color coded to easily identify the different regions. Before describing the different sections of the model it is important to note that the model is created from the view of the planner which is always the row player. In saying this the model works to help the row player make a decision while anticipating uncertainty in the column players strategy and assessing the risk of making certain decisions in this uncertainty. In the green/bottom right region of the model, is the basic two player game theory normal form model. The m strategies of the row player are placed against the n strategies of the column player. The utilities for each player when certain strategies are played are also recorded here as normal. For example, if the row player chose strategy row 2 and the column player chose col 1 the utility for each player would be found in the box u_{21}. This is where the similarities to standard game theory normal form end as the other sections are unique to HNF. The next section to look at is the orange/upper right region of the model. This section captures the column mixed strategies (CMS). Similarly the blue/bottom left region captures the row mixed strategies (RMS). Each of the k rows, labeled $C_{strategy\#}$, contain a mixed strategy of the column player with the values in each row summing to 1. This can be represented mathematically as $\Sigma_{j=1}^{n} c_{kj} = 1$. Each same numbered CMS, $C_{strategy\#}$, corresponds to the same numbered RMS, $R_{strategy\#}$. The first CMS,

C_0, always contains the Nash equilibrium mixed strategy (NEMS) of the full game for the column player as seen in Table 9

For example, $c_{0strategy\#}$ would contain the probability that the column player should use the corresponding col (strategy #) strategy according to the calculated NEMS. CMS C_0's corresponding RMS, R_0, contains the probability values of its strategies according to the full game NEMS as well. Just like in the CMS, the RMS components must sum to 1 which mathematically means $\Sigma_{i=1}^{m} r_{ki} = 1$. There is an implicit and explicit way to determine the other CMS, C_k where $k \neq 0$. For the implicit CMS the row player believes that the column player is playing a $u \times v$ subgame of the full $m \times n$ game where $1 \leq u < m$ and $1 \leq v < n$. When this is the case a '-' is placed in the location of the CMS's strategy that is not being considered and same goes for the RMS. After that the NEMS is calculated for the subgame and the probability values are placed in the strategies still being considered. The explicit CMS is found when the row player believes the column player is playing their strategies at pre-determined probabilities. For this CMS, values of 0 are entered for strategies that are played with a 0 probability and also the corresponding RMS contains only '-'since the CMS is not guaranteed to be at NEMS. The top row of the CMS section is labeled C_Σ. Here the mixed strategy of the entire games is recorded. This is where the yellow/top left corner of the model comes into play. This section is known as the belief context and each value P_k represents the percentage the row player believes the column player is playing CMS C_k. For example, if the row player believes that the column player is 80% likely to be playing the full game then P_0=0.8. The sum of all the belief contexts must also sum to 1, so this condition must hold, $\Sigma_{k=0}^{K-1} P_{k=1}$. In order to calculate C_Σ, multiply each strategy in each CMS by its belief

Table 9. C_0

C_0	c_{01}	c_{02}	c_{03}	...	c_{0n}

context associated with that CMS and then that value is summed together with the like strategies across all the CMS to get the cumulative CMS of the hypergame. The equation for this is $S_j = \Sigma_{k=0}^{K-1} c_{kj}$. Once again the cumulative CMS, C_Σ, sums to 1 so that $\Sigma_{j=1}^{n} S_n = 1$. Table 10 puts the game of Rock, Paper, Scissors into a hypergame to demonstrate these concepts.

It is seen the strategies and utilities are the same as they were in the standard game theory normal form. Also the NEMS for the full game is contained in row C_0 and column R_0. The CMS for C_1 represents an implicit subgame. In this subgame the row player believes the column player is playing a game where they are not allowed to play scissors, for whatever reason. This is represented by the '' in the scissor strategy for the C_1 CMS. Since it is an implicit CMS the row player is able to calculate their NEMS strategy for that game as well which is in the column R_1. The benefit of this observation is that the Row player may be able to trick the column player into thinking that certain options are unavailable which could pay off for the row player. Although this may be hard to do in Rock, Paper, Scissors the benefits can be easily seen in a warfare scenario where an opponent might make certain strategies seem unavailable to their opponent like in the Fall of France. The CMS for C_2 is an explicit strategy and represents the situation where the row player believes the column player only chooses to play the Rock strategy even though the column player knows the other strategies are available. The row player does not record a RMS in this situation which

Table 10. Rock, Paper, Scissors Hypergame Normal Form

			C_Σ	0.533	0.3	0.167
.3			C_2	1	0	0
	0.2		C_1	1/3	2/3	-
		0.5	C_0	1/3	1/3	1/3
R_2	R_1	R_0=full game	row\col	Rock	Paper	Scissors
–	0	1/3	Rock	(0, 0)	(-1,1)	(1, -1)
–	2/3	1/3	Paper	(1, -1)	(0,0)	(-1,1)
–	1/3	1/3	Scissors	(-1,1)	(1, -1)	(0,0)

can be seen by '-'s in column R_2. The belief contexts show that the row player has 50% belief the column player is playing the full game, 20% belief the subgame, C_1, is being played and 30% belief the explicit game, C_3, is being played. The values for the cumulative mixed strategies are

$$S_1 = 1/3 * 0.5 + 1/3 * 0.2 + 1 * 0.3 = 0.533$$
$$S_2 = 1/3 * 0.5 + 2/3 * 0.2 + 0 * 0.3 = 0.3$$
$$S_3 = 1/3 * 0.5 + 0 * 0.2 + 0 * 0.3 = 0.167$$

Now that these values have been established, it's time to talk about the 4 different strategies that the HNF allows us to look at. When deciding which strategy to ultimately choose, risk and fear of being out-guessed is taken into consideration.

The benefits of the HNF is that it allows a planner to look at the game in different ways and directly compare these different approaches against each other. Vane proposed 4 unique approaches that a planner can choose from, deemed hyperstrategies, which are simply four different ways to formulate a mixed strategy. These four hyperstrategies are the NEMS, Modeling Opponent (MO), Weighted Subgame (WS), and the Pick Subgame (PS) strategy. These strategies are compared to each other by their EU. The EUs of the hyperstrategies are based off the cumulative mixed strategy for the column player C_σ. Since the planner may be unsure of their perception in the opponents strategy, the EU of a given hyperstrategy for the planner changes, depending on how much confidence the planner has in C_σ. This can be seen as the row player's confidence the belief contexts he assigned the column player's different perceptions. Vane characterized this measure of confidence as a fear of being out-guessed, where having less fear is equal to having more confidence. These concepts are explained further in the following sections. The first hyperstrategy to look at is the NEMS. This hyperstrategy is the basic mixed strategy found when solving the Nash equilibrium of the full game. Once this mixed strategy for the row player is

found, the EU for the NEMS game can be found as well as the hyperstrategy EU (HEU) when there is belief in subgames and explicit games being played. The EU from belief in these other games is found by considering the column players mixed strategy to be C_Σ as opposed to C_0 for the regular EU of the full game. Table 11 shows the Rock, Paper Scissors Game with additional rows on the bottom of the RMS section which are used to record these EU values.

The EU is calculated by multiplying the utility of a cell by the chance it has of being selected which is found by multiplying the corresponding RMS and CMS probabilities together. This is represented by the equation $EU = \Sigma_(i = 0)^m \Sigma_(j = 0)^n u_{ij} * c_{kj} * r_{lj}$ where k refers to a CMS to include C_Σ and l refers to a specific hyperstrategy. The calculations for these values turn out to be 0 for both.

$$EU(*, C_\Sigma) = 0 * 0.533 * 1/3 + (-1) * 0.3 * 1/3 + 1 * 0.167 * 1/3 + 1 * 0.533 * 1/3 + 0 * 0.3 * 1/3 + (-1) * 0.167 * 1/3 + (-1) * 0.533 * 1/3 + 1 * 0.3 * 1/3 + 0 * 0.167 * 1/3 = 0$$

$$EU(*, C_0) = 0 * 1/3 * 1/3 + (-1) * 1/3 * 1/3 + 1 * 1/3 * 1/3 + 1 * 1/3 * 1/3 + 0 * 1/3 * 1/3 + (-1) * 1/3 * 1/3 + (-1) * 1/3 * 1/3 + 1 * 1/3 * 1/3 + 0 * 1/3 * 1/3 = 0$$

Table 11. Rock, Paper, Scissors NEMS

			C_Σ	0.533	0.3	0.167
0.3			C_2	1	0	0
	0.2		C_1	1/3	2/3	–
		0.5	C_0	1/3	1/3	1/3
R_2	R_1	NEMS 3x3 game	row\col	Rock	Paper	Scissors
–	0	1/3	Rock	(0, 0)	(-1,1)	(1, -1)
–	2/3	1/3	Paper	(1, -1)	(0,0)	(-1,1)
–	1/3	1/3	Scissors	(-1,1)	(1, -1)	(0,0)
		0	EU(*,C_Σ)			
		0	EU(*,C_0)			

The next hyperstrategy is MO. In this hyperstrategy the planner plays the pure strategy or row that gives the best EU with consideration to C_Σ while the other strategies are given a probability of 0. The EUs are calculated the same way as for the NEMS. The zero-sum hypergame in Table 12 demonstrates how this is calculated. There is an additional column in the RMS section for the MO mixed strategy.

The EU for each row is calculated and shown in parentheses in the MO column.

$$row\ 1\ EU = 2 * 0.5165 + 5 * 0.2167 + -3 * 0.2668 = 1.3161$$

$$row\ 2\ EU = 3 * 0.5165 + 0 * 0.2167 + 1 * 0.2668 = 1.8163$$

$$row\ 3\ EU = -1 * 0.5165 + 6 * 0.2167 + -2 * 0.2668 = 0.2501$$

Since row 2 has the highest EU it is chosen as the MO strategy which can be seen by the '1' in row 2 of the MO column. MO has an equal or greater EU than the NEMS in both EU (*,C_0) and EU (*,C_Σ). Therefore the row player can only gain by playing the MO hyperstrategy. This shows how a planner can use the belief that the opponent is possibly playing different games to their advantage and make decisions with potentially higher payoffs. Vane refers to strategies like this as fully effective because they are no worse than NEMS while a partially effective hyperstrategy can potentially do worse.

Table 12. Hyperstrategy MO

				C_Σ	0.5165	0.2167	0.2668
	0.1			C_2	1	0	0
		0.5		C_1	0.833	0.167	–
			0.4	C_0	0	0.333	0.667
MO	R_2	R_1	NEMS 3*3 game	row\col	col 1	col 2	col 3
0(1.3411)	–	0.5	0	row 1	2	5	-3
1(1.8163)	–	0.5	0.889	row 2	3	0	1
0(0.2501)	–	0	0.111	row 3	-1	6	-2
1.8163			1.642	EU(*,C_Σ)			
0.667			0.667	EU(*,C_0			

The next hyperstrategy is WS. In this scenario the planner demonstrates his belief that subgames are being played and places his weight on each subgame proportionally to the belief context. To find the mixed strategy values for WS, each subgames mixed strategy is multiplied by their respective belief context. The resulting values are summed for every strategy to obtain the mixed strategy probability for a particular strategy. The equation for the WS mixed strategy is $ws_i = \Sigma_{j=0}^{k} r_{ij} * P_i$ where i is a row for a particular strategy and j is a particular RMS. The normal form hypergame seen in Table 13 gives an example of the WS.

The probability calculations for the WS column are:

$$Row\ 1 = 0.5 * 0.3 + 0 * 0.7 = 0.15$$

$$Row\ 2 = 0.5 * 0.3 + 0.889 * 0.7 = 0.7723$$

$$Row\ 3 = 0 * 0.3 + 0.111 * 0.7 = 0.0777$$

In this case the EUs for the WS were less than that of the NEMS meaning the WS would not be an effective hyperstrategy. This is not always the case however as WS can be effective in other games given the right conditions.

The last hyperstrategy that remains to be discussed is PS. PS is fairly simple. This strategy requires the planner to choose the subgame with the least amount of columns. Also no matter what the amount of rows in the subgame, all the rows are

Table 13. Hyperstrategy WS

			C_Σ	0.25	0.2831	0.4669
	0.3		C_1	0.833	0.167	–
		0.7	C_0	0	0.333	0.667
WS	R_1	NEMS 3*3 game	row\col	col 1	col 2	col 3
0.15	0.5	0	row 1	2	5	-3
0.7723	0.5	0.889	row 2	3	0	1
0.0777	0	0.111	row 3	-1	6	-2
1.057		1.139	EU(*,C_Σ)			
0.5163		0.667	EU(*,C_0			

included when calculating the EUs. This is because having more rows for the planner can only increase the EU. Table 14 is an example of the PS in the HNF.

For the PS the C_1 CMS was chosen as it has the least amount of columns. The results show both EUs for PS being lower than that of the NEMS making this strategy ineffective.

Another way to gauge the effectiveness of a strategy is to compare the hyper EU to the worst case scenario utility for a which Vane refers to as G. G is found by finding only the worst column the row player can go against and calculating that EU. Table 15 shows the Rock, Paper Scissors Game with additional rows on the bottom of the RMS section which are used to record these EU values.

To find the G value for a mixed strategy, calculate the EU against each column. So for PS, find the EUs when column plays Rock, Paper and Scissors.

$$Rock\ EU = 0 * 0 + 2/3 * 1 + 1/3 * -1 = 1/3$$

$$Paper\ EU = 0 * (-1) + 2/3 * 0 + 1/3 * 1 = 1/3$$

$$Scissors\ EU = 0 * 1 + 2/3 * (-1) + 1/3 * 0 = -2/3$$

The worst case for the PS strategy occurs when column plays scissors and the value of -2/3 is stored in G. The Rock, Paper Scissors normal form hypergame in Table 15, shows that the PS has a higher EU when considering subgames than the NEMS and that the EU for C_0 is equal. This may make it seem like PS is a full proof strategy.

Table 14. Hyperstrategy PS

			C_Σ	0.25	0.2831	0.4669
	0.3		C_1	0.833	0.167	–
		0.7	C_0	0	0.333	0.667
PS	R_1	NEMS 3*3 game	row\col	col 1	col 2	col 3
0.5	0.5	0	row 1	2	5	-3
0.5	0.5	0.889	row 2	3	0	1
0	0	0.111	row 3	-1	6	-2
0.8658		1.139	EU(*,C_Σ)			
0.1655		0.667	EU(*,C_0)			

Table 15. Hyperstrategy with G Values

			C_Σ	0.333	0.467	0.2
	0.4		C_1	1/3	2/3	–
		0.6	C_0	1/3	1/3	1/3
PS	R_1	NEMS 4*4 game	row\col	Rock	Paper	Scissors
0	0	1/3	Rock	(0,0)	(-1,1)	(1,-1)
2/3	2/3	1/3	Paper	(1,-1)	(0,0)	(-1,1)
1/3	1/3	1/3	Scissors	(-1,1)	(1,-1)	(0,0)
0.133		0	EU(*,C_Σ			
0		0	EU(*,C_0			
-0.667		0	G			

However, when looking at the worst case scenarios, it is seen that PS does worse with an EU of -0.667 compared to 0 for NEMS. This shows how the HNF can allow a planner to gauge the risk they are making in their decisions and find strategies that can take advantage of knowledge gained against the opponent. The variable g is used to quantify how much a planner believes he may be outguessed. The variable g can take on a value from 0 to 1 representing the percentage of this belief. 0 means that the planner has no fear of being outguessed while a 1 indicates the planner believes 100% he is being outguessed and all belief should be only in the full game. The g value is used to calculate the HEU of any hyperstrategy. The equation is $HEU(hyperstratgey) = EU(hyperstrategy, C_\Sigma) - g * (EU(hyperstrategy, C_\Sigma) - G)$ This equation produces a value that is between EU(hyperstrategy,C_Σ) and G based on the value of g. These values represent the best and worst case values. These values can also be seen as the EUs with no fear of being outguessed and complete fear. Once a planner determines their g value, they can determine their HEU and make decisions based off this value.

2.4 Cyber Warfare

Since its introduction hypergame theory has been applied to many fields to better understand interactions and outcomes in conflicts and gather insight for future planning. Cyber warfare is one of those fields that has been explored when it comes to hypergame theory although this research is still considered in its infancy. Game theory has been used in the past to consider cyber warfare scenarios and a lot of insight has been gained from it. Usually these games involve an attacker and a defender of a network, both selecting different strategies to maximize their utilities. However, cyber warfare deals a lot with deception and learned belief about an adversary in the real world. Hypergame theory is prime to model these aspects of cyber warfare where simple game theory may have trouble. Using hypergame theory the planner can make more dynamic decisions based on their perceptions of the adversary which could lead to better outcomes. In order to understand how to model these situations it is best to first understand physical systems. This section briefly touches on CPS to better understand the models.

Cyber-Physical Systems.

Ever since the computers have been used to process and pass information quickly and efficiently, people have sought ways to harness this capability to improve processes in industry as well as products that affect our daily lives. This integration of the cyber world with tangible physical systems creates what is known as CPS. A CPS is a system that utilizes computer technology to affect a physical component in the real world. Figure 4 [36] shows an example of the basic features of a CPS. First are the computers that provide the command and control of the system. They pass information over communication medium that may be directly wired or connected over a wireless network to actuators. Actuators are mechanisms, such as motors,

that are able to take input from the computer through a digital to analog converter (DAC) and perform the specified task. On the other end of the system are the sensors that capture information from the physical world and use analog to digital converters (ADC) to send information to the computers. The computer takes that information and decides how to affect the actuators and continue the feedback loop. Real world examples of these systems can include power plants that utilize supervisory control and data acquisition (SCADA) control systems, smart vehicles that have built in computers that make it easier and safer to drive, pace makers that control a patient's heart rate, and even future talks of smart cities where the city's infrastructure is monitored by sensors and maintained and controlled by a network of computers [37]. From these examples alone there is no question why it is important these systems function properly. As cyber technology is increasingly integrated into physical systems in today's world, the stakes for securing these systems increase as well.

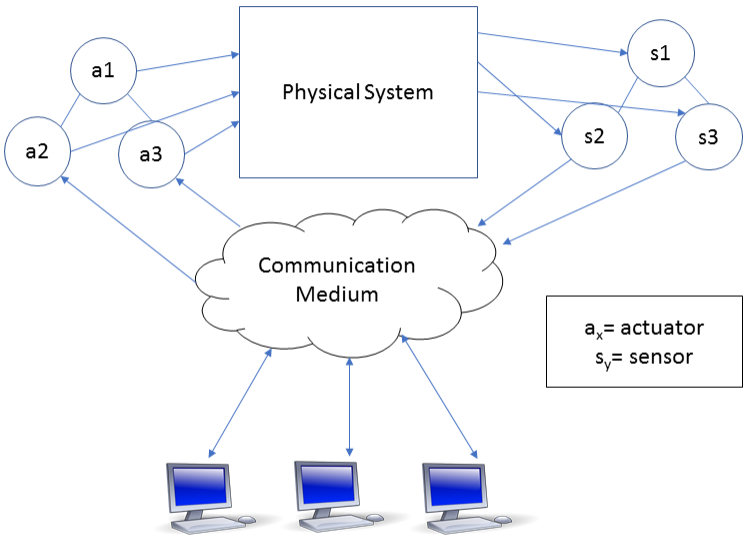

Figure 4. Basic Cyber-Physical System

Security.

As cyber technology continues to proliferate and integrate into different physical systems these systems can become vulnerable to cyber attacks. Prior to the advent of cyber integration, attackers would need to physically interact with a system to achieve their desired outcome. This posed several risks on the attacker including the risk of getting hurt and greater chance of being caught. Through cyber attacks the attacker can remain out of harm's way and mask their true identities to prevent attribution. These cyber attacks target the channels of communication as well as the communications themselves to disrupt system capabilities and/or intercept data. A secure CPS must be able to maintain the traditional cyber security goals of integrity, availability, and confidentiality. Integrity in CPS refers to the ability to ensure communication is being passed throughout the system without being altered. A cyber attack targeting communication integrity in a CPS can cause system delays, improper functions and even system failure. Availability in CPS refers to the ability for communications flow freely at all times. Although a cyber attacker can reduce the availability of a CPS by affecting the integrity of the communications, another way to accomplish this is by simply targeting the communication medium to prevent information from being passed properly in the first place. Lastly, confidentiality in regards to a CPS refers to protecting information that may be proprietary, or private from getting released. One way to accomplish this is through utilizing secure channels of communication and secure protocols. One form of defense can help reach multiple cyber security goals other than their specific intent as seen with secure channels of communication. It is also key to point out that security mechanisms in a CPS can run concurrently so the defender can defend all the potential threats at the same time which is also within their best interest although a lot cyber models only allow a single defense at a time.

2.5 Modeling Cyber Systems via Game Theory

New kinds of cyber attacks and ways to defend against them are constantly being encountered so it should be no wonder that there have been numerous models built to capture the different aspects of these scenarios. Given their inherent differences these models do share some similarities, one being that models simplify reality. Real life cyber attacks can be very complicated with the multitude of attacks, methods of defense, and potential targets. Researchers have attempted to compile the different kinds of scenarios in cyber that game theory can be applied to [38]. Attempts to take into account all the different variables and present the results in a sensible fashion would be a very arduous task even for a computer. While the following discussed models look at their cyber situations with varying degrees of detail they mainly focus on the key aspects of the attack and defense. This focus allows efficient analysis of the situation while still providing some meaningful insight that can be applied in more realistic conditions. Another feature that these models share are having similar participants. These players include defenders and attackers pitted against each other commonly, although not always, yielding a two player game. During real cyber attacks there can be more than two actors at play, but the reasoning for predominantly two player models goes back to the idea of simplification. The actual differences between models lies in a few key areas. These areas include the actions available to the game participants, how utility are calculated, the scenario, and lastly the structure. By varying these elements most cyber scenarios can be modeled, with each variation providing their own benefits. The following text considers some of the types of models that can be created by changing or even combining the game types.

Linear Programming.

Although a linear programming model is not regarded as typical game model system it can be very effective at modeling a game where each player is given objectives and constraints. From a game perspective however the goal is not to maximize each player's payoffs, but to find a Nash equilibrium where each player has no incentive to deviate from their actions given the other players' actions. Simply optimizing actions to maximize each player's objectives would instead produce a Pareto front of solutions. In a Pareto front no player would be able to change their action without possibly decreasing the utility of the other players, which ii a zero-sum game implies incentive for a player to deviate from their actions and receive a better pay off. One example of linear programming used to model a two player cyber defense game is seen in Ming Zhang's research on stealthy attackers on a network [39]. Ming Zhang effectively used linear programming to model how a defender would protect networked nodes from an attacker who is knowledgeable on the environment and is also trying to remain undetected as it attacks different nodes. Using the notations in Table 16linear programming model displayed in Equations 3 and 4, Zhang was able to find the conditions necessary for a Nash equilibrium and conjectured there could potentially be more than one. Equation 3 has an optimization function that is maximizing the sum of the defender's payoffs when the nodes are compromised. In this equation, $p_i r_i$ represents the benefit to the attacker. Meanwhile $m_i(r_i w_i p_i - C_i^D)$ is the total benefit to the defender after subtracting the cost of the defense for that node. The value is negated because the less benefit the attacker receives the better for the defender. This could have been avoided by having the objective be minimized although the current set up essentially works the same. The constraints show that the defender's frequency of recovery on the nodes is limited by their budget and the attacking time on the node. The attacker's Equation 4 is similar to Equation 3. The attacker is

trying to maximize their benefit found in r_i. However the attacker must also factor in the cost to attack a node and the defender's efforts to recover the node captured in $m_i(r_i w_i - C_i^A)$. The attacker's constraints show that they are limited by their budget, and the sum of their efforts distributed over the nodes cannot exceed 100%. This method of modeling network security incidents works when each player's action are more easily represented by variables with changing values and when constraints are being considered.

Table 16. Linear Program Model Notations

Symbol	Meaning
N	number of independent nodes
r_i	benefit per unit of time by compromising node i
w_i	attacking time for node i
C_i^A	attackers move cost for node i
C_i^D	defenders move cost for node i
B	budget to the defender
M	budget to the attacker
m_i	frequency of recovery action for node i
p_i	probability of immediate attack on node i once it recovers

Defender's Modeled Objective

$$\begin{aligned} \max_{m_i} & \sum_{i=1}^{N} -[p_i r_i - m_i(r_i w_i p_i - C_i^D)] \\ \text{s.t.} & \sum_{i=1}^{N} m_i \leq B \\ & 0 \leq m_i \leq \frac{1}{w_i}, \forall i \end{aligned} \tag{3}$$

Attacker's Modeled Objective

$$\begin{aligned} &\max_{p_i} \sum_{i=1}^{N} p_i[r_i - m_i(r_i w_i - C_i^A)] \\ &\text{s.t.} \sum_{i=1}^{N} m_i w_i p_i \leq M \\ &0 \leq p_i \leq 1, \forall i \end{aligned} \tag{4}$$

Normal Strategic Form.

One of the most popular game types used to model situations in various fields is the normal strategic form game. Used in many classic games in game theory, this model features actions for the attacker and defender and payoffs based off every possible combination of those actions. An example of this model in cyber defense can be found in the model proposed by Anibal Sanjab and Walid Saad [40]. From this research, in Table 18 is the zero sum normal form game and Table 17 the notations used to model an incident where a defender must choose which cyber assets on a network to protect from an attacker who can have varying levels of ability unknown to the defender. The research was able to show that a defender could benefit from making moves that deviated from the equilibrium if the attacker was not operating at a high level. Strategic form games benefit from being simple to understand and build, however they operate with the assumption that a defender and an attacker make moves simultaneously which is not always the case.

Table 17. Normal Form Model Notations

Symbol	Meaning
N_c	total number of cyber nodes
N_p	total number of physical nodes
C	set of cyber nodes in system
P	set of physical nodes in system
$r_{c,p}$	probability cyber node $c \in C$ failed, given physical node $p \in P$ failed
κ_c	probability of failure of c, $\kappa_c = 1$ if c not defended from attack otherwise $\kappa_c \approx 0$
π_p	probability of failure of p, $\sum_{c=1}^{N_c} r_{c,p}\kappa_c$
f_p	cost of failure of p
n_a	total number of nodes that can be attacked at any given time
n_d	total number of nodes that can defended at any given time
S_i	set of strategies s_i, $i \in \{a, d\}$, s_i is a subset of cyber nodes, N_c, being attacked(s_a)/defended(s_d), $\lvert S_i \rvert = \binom{N_c}{n_i}$
E_f	expected total loss to the system, $E_f = \sum_{p=1}^{N_p} \pi_p f_p$, $U_d = -E_f$ and $U_a = E_f$

Table 18. Normal Form Game

Attacker/Defender	Actions = s_j^a
Actions = s_i^d	Utility =$(U_d(s_i^d, s_j^a), U_a(s_i^d, s_j^a))$

Stackleberg Game.

There are times when a defender may make a move first and the attacker responds and vice versa. This can be seen in the game studied by Lin Chen where cyber assets armed with intrusion detection systems are protected and attacked with different probabilities. The scenario was modeled with a strategic form game first and then a Stackelberg game [41]. A Stackelberg game is where one of the game's players makes the first action and the other players react to it. This style of game facilitates the analysis of scenarios where a defender can see the strategy of an attacker and react accordingly and vice versa. Another example of a Stackelberg game can be seen in the research model of a network administrator applying patches to affected systems after an attacker has introduced viruses into the environment [42]. Using a Stackelberg

game is just one way of modeling a scenario where the actions are not simultaneous as they can also be modeled using the extensive form. Similar to extensive form games, the Nash equilibrium of a Stackelberg game can be solved through backwards induction as seen in the earlier example. The additional benefit of an extensive form game is that it works well at capturing multiple rounds of interactions and can easily be converted to strategic form if needed

Signaling Game.

Another game type that allows players to take turns is the signaling game. The signaling game is very similar in purpose to the Stackelberg game but it builds on the idea of uncertainty. In a signaling game there are two players which consist of the sender and the receiver. The sender has the property that they can be one of several types. Different types of senders have access to different actions which are referred to as messages. After the sender completes his action/sends his message, the receiver receives it and depending on the action they receive, they complete their own action that gives them the best payoff. The sender also receives a payoff based on the receiver's final actions. The uncertainty in the game stems from the fact that the receiver is unaware of the type of sender, which factors into the payoffs each player gets after the receiver's actions. Therefore it is up to the receiver to make the best determination of the sender's type based off the message they receive from the sender. This type of game allows for the sender to mask their potential payoffs to the receiver as well as their future options for messages to the receiver if there is subsequent play. This game type has been used to model a network of nodes where a defender, playing the receiver, is responsible for protecting the good IDS equipped nodes from the malicious nodes [43] and the notations for this model are seen in Figure 19. The uncertainty lies in the defender not knowing the true

identity of the malicious nodes, therefore having to react to the messages they receive from the sender. Another example of this game can be seen in the 2004 paper by Animesh Patcha which applied game theory to intrusion detection in mobile ad hoc environments. A similar approach was used by Hayreddin Ceker in 2016 [44]. This research looked at the scenario where a network is being subject to denial of service (DOS) attacks. As can be seen in figure 5, the defender acting as the sender sends a signal to the attacker attempting to mask whether the sender is a normal system or actually a honey pot that wants to be attacked. The values by the attacker's decision in the diagram represent the utility awarded to the defender and attacker respectively. By using a proper strategy the defender can convince the attacker to target the honey pot which they can then use to analyze the attacker's strategies and improve defenses.

Table 19. Signal Game Model Notations

Symbol	Meaning
A	attacker (signal reciever)
D	defender (signal sender)
θ_D	nature's decision of defender type
α^N, α^H	probability of signaling 'N' from normal type and honeypot defender, respectively
μ	attacker's belief that the 'N' signal is received from normal type defender, $1-\mu$ from type honeypot
γ	attacker's belief that the 'H' signal is received from normal type defender, $1-\gamma$ from type honeypot
c_a, c_o	attacker's cost of attacking and observing respectfully
b_a, b_o	benefit of attacking and obesrving respecftfully
c_c, c_s, c_h, c_w	defender's cost of compromise, signaling, honeypot and being watched, respectively
b_{cs}, b_w	customer satisfaction on a normal system and benefit of watching an attacker on a honeypot respectively

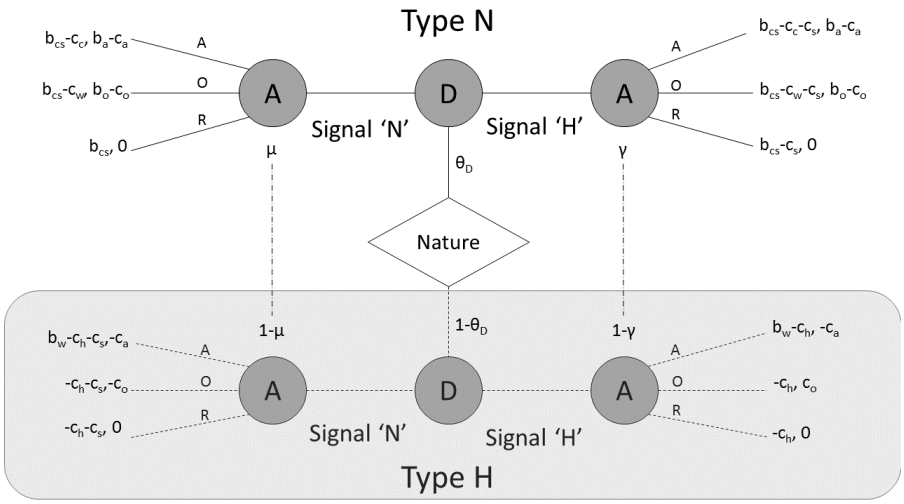

Figure 5. Signal Game in Extensive Form

Stochastic Game.

Another game type that can be used to model cyber scenarios with uncertainty is the stochastic game. The stochastic game works by having both players select an action just like in a normal strategic game. Depending on the actions selected by the players, the actors transition to a different state and receive their payoffs. The state the players transition to is not deterministic however, but probabilistic based off the prior state and the actions selected by the players. This game type is very good at modeling situations where the environment is not completely known or where the same actions do not necessarily guarantee the outcomes. Also stochastic games are very useful for modeling games where there is repeated play as the actors can transition to multiple states and receive a final payoff based off the payoffs received throughout the game. In the paper "Dynamic Policy Based IDS" by Quanyan Zhu [45], the stochastic game was used to model how an intrusion detection system would respond to different attacks it receives. The notations for this model are seen in Table 20. As the attacker would send different attacks the IDS would respond by switching to different protection configurations which would place the computer system in different

states as seen in the figure below 6. The research was able to show that a dynamic IDS capable of wisely changing its level of protection dependent on the situation would operate more efficiently than a typical static IDS systems while still providing acceptable security.

Table 20. Stochastic Game Model Notations

Symbol	Meaning
S	set of possible states, s_i, of the computer system
L	set of available libraries, l_j, that the detector can use to detect attacks
F_i	the detectors configuration which is a subset of the available libraries L
c_i	cost of a particular library l_i
a_i	an attack available to the attacker
d_i	damage caused by a particular attack a_i given the current state and detector configuration, F_i
r	utility received by the players which is dependent on the current state and actions of both players

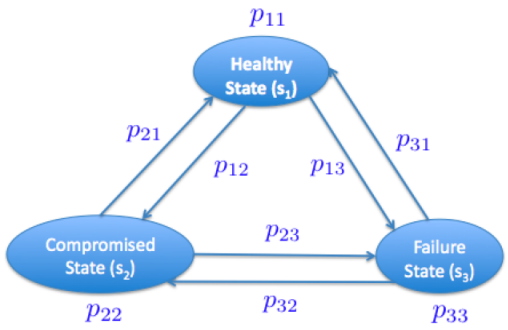

Figure 6. Stochastic Game Flow Chart

Hypergames.

So far the models presented were analyzed without using hypergame analysis. Hypergame analysis is beneficial because it examines games where players make decisions based off their perceptions of the game which can be affected by different factors that are overlooked in traditional game theory. The analysis of these games can possibly lead to more accurate or realistic conclusions for the decision maker. This allows

different players to be playing different versions of the same full game. The different versions of the game are created when certain aspects of the full game are removed from a player's knowledge creating an incomplete information game. Players may be unaware of certain actions available to the other players or even unaware of other player's in general. If a player is capable of keeping aspects of the full game hidden to the other players they may be able to use the information to gain an advantage and a more favorable outcome. A detailed discussion of the hypergame theory model can be discovered in Kovach's 2015 paper [11].

This idea of hypergames was used in 2016 to model the interactions between a network administrator and an insider threat [46]. This paper focused on getting readers spun up on hypergame theory but it also gave a game example of how it could be used. In this game the system administrator had access to stratagem actions that the insider threat was unaware of. The stratagem acted similar to a honeypot. The system administrator was able to use this action to deceive the insider and get a better outcome then previously obtainable. The paper also explored the use of deception in iterative play and how a system administrator would have to adapt to an evolving insider threat.

Another way to model hypergames is through the normal form hypergame discussed earlier in this chapter. As stated in previous sections the uncertainty comes from the row player having to form a probabilistic distribution over which sub-game the column player is participating in which determines which actions are available to the column player. This allows a multitude of different situations to be modeled when a defender is unsure of the attacker or vice versa. This can be seen in recent research where a hypergame model was used to examine how a defender would be able to adapt their beliefs of an attacker's true strategy based on the attacker's actions. This research showed that a defender that altered their belief of an opponent's skill in

repeated encounters could better protect the network at a lesser cost. The strengths of the hypergame model also showed how a defender and an uncertain attacker would react to each other after iterated play [10]. This model is unique in that the attacker was actually made the row player. This allowed the analysis to capture the uncertainty an attacker has for a defender's capabilities, thus more closely modeling an aspect of real life network interactions. This also highlights how in a hypergame, either player can be used to as the row player to get additional insight into the uncertainty of a cyber situation. Additionally the author researched and demonstrated it is possible for a player to purposefully act in ways to skew another player's beliefs in order to gain a better outcome in future interactions. This demonstrated how a hypergame model can also be used to consider deception amongst players.

In 2013 Yadigar developed a hypergame model for information security as a way to address real world situations where the players had asymmetric perceptions of the game they were playing [47]. He proposed a two level hypergame based off Bennet's model and used the HYPANT hypergame software to run his experiments. Through this effort Yadigar was able to demonstrate numerically how hypergames can be used to select the best attack and defense actions in a contested network.

Hypergame theory was also used to model an attacker-defender game from the point of view of the defender by Alan Gibson in his paper Cyber Security using Hypergame Theory [9]. In the paper Gibson uses Vanes HNF to run games with different levels of attackers and defenders. The different levels for both attackers and defenders included All Out, High-Level, Mid-Level, Low Level and Nuisance Defender. These different levels were used to update the functional utility values which were influenced by the level of the player. The utility values were calculated in such a way that the defenders received the best utility when going against a similar level attacker. This means, that an All-Out defender would receive the best utility against

an All-Out attacker. Against any other attacker an All-Out defender would be over-reacting which would cost the defender. Each combination of level of attacker and defender was placed into a HNF model. Each game was also played over multiple iterations. These iterations included all values being static, only the variables being updated, only the belief contexts being updated, only the g values being updated, and all values being updated. The purpose of this set-up was to compare the results of an attacker defender model that was static to a more realistic one where a defender and attacker would make adjustments and change their beliefs based on events of a previous game, basically learning from one game to the next. The results of this paper showed that the defender can increase their utility ratio by using information gained from previous games. The results show that the defender was able to get a better utility when adjusting their belief contexts and g value in response to past games. This demonstrates how experience can be used to build good belief contexts that can result in better decision making for the planner. This section showcased some of the few examples of hypergame theory being used in a cyberspace scenario so far and there remains a lot more that can be explored in this space.

2.6 Chapter Summary

This chapter introduces the history of game theory being developed from decision theory. Hypergame theory expands on game theory by accounting for misperceptions player's may have in a game. This allows for a more accurate modeling of different game scenarios. It also supports the idea of how a player can deceptively incite misconceptions into the other players to achieve payoffs that are greater than the Nash equilibrium. The HNF is able to capture the payoffs players could expect given their beliefs on the other player and scale these values according to the player's fear of being outguessed. CPSs are the integration of computers with physical components

and are therefore vulnerable to cyber attacks. Game theory has been used to explore the interactions between a defender and an attacker in a cyber attack. Hypergame theory is also utilized to more accurately model the deception and misperceptions that are common in these cyber scenarios. This sets the stage to look at a unique attacker/defender model based on a CPS developed in the next chapter and the methods that are used to analyze it.

III. Methodology for Hypergame Analysis

In a cyber defense scenario there are a lot of different factors that come into play. These include the actions available to the defender as well the attacker, and how these different actions interact with each other to produce interesting outcomes and payoffs to the participants. This chapter introduces a modeling framework and usinf that framework, sees the development of a model that can be used to investigate the behaviors of a defender and an attacker in a cyber defense scenario for a cyber-physical system (CPS). The benefits of the hypergame normal form (HNF) method of analysis are elaborated on. This chapter also discusses aspects of AFIT's Hypergame Analysis Tool's (HAT) features, including recent upgrades and how they are used to conduct model analysis.

3.1 Model

In order to analyze cyber conflicts, a model must be created that acts as a representation of a real life scenario. Although the model is not expected to contain all the complex interactions of an actual event it should maintain the key elements so that the analysis can carry over information to the real world. In this section a cyber defense modeling framework (CDMF) is introduced that simplifies the development of a cyber defense model, and is flexible enough to model a multitude of cyber scenarios and environments. The CDMF is used to develop a model involving cyber conflict in a CPS for analysis in the HAT software.

The CDMF first finds the actors that participate in the game. Typically the actors are going to be a defender, such as system administrator, and an attacker. Once the actors are determined, the possible actions of the attacker can be listed. For each action of the attacker, the defender should have a counter action. Once the actions are

determined, its time to find the payoffs for each possible pair of the attacker's actions with the defender's actions. When calculating the payoffs it is important to take into account the cost an action has on the actor to perform the action. The payoffs also take into account the accomplishment of the mission at hand. The attacker's goal is to achieve some minimum level of damage and the defender's goal is minimize the level of damage. In this framework, the attacker can receive a penalty if the mission is not met. The attacker does not ever get a bonus for achieving the mission because the bonus is already captured in the damage done. In a similar vein the defender can receives a bonus for completing the mission at hand. The defender does not have a penalty applied to their payoffs if they fail the mission because this is captured in the damage done by the attacker. Once the payoffs are determined a normal form game can be built using each actor's actions along with the payoffs found. The benefits of this model are that it is simple, straightforward, and can easily be applied to a normal form game for quick analysis. Other researches have used this same approach [10], although this is the first time this framework has been formalized. Other frameworks have been proposed to model cyber network scenarios [41], but the CDMF framework focuses on being simple and flexible while other frameworks proposed are more complex and can be limited by their structure. Another unique benefit of this framework is that it incorporates the idea of completing or failing a mission into the payoffs which is absent in other models. This is important to capture because it provides incentive for the attacker to avoid inaction and also for the defender to protect their system despite experiencing a cost.

The model that is used in the experiment is based off possible threats present in a wireless sensor network (WSN) of a CPS [48]. Due to the wireless nature of this network it is much more exposed to adversarial tampering since communications are broadcast and anyone can be listening. Therefore it is important that a model

can capture this scenario and be analyzed to help prepare a defender of this network for these unique challenges. The first part of developing the model for the CDMF framework is to define the actors. In this case the actors are the system administrator, referred to as the defender, and the adversary targeting the WSN, known as the attacker. Once the actors are determined their actions must be defined starting with the attacker. In this model 4 actions have been identified that the attacker can perform . The first is eavesdropping which constitutes the attacker listening in on communications that are being passed between the wireless sensors in order to gain private information. This action however does not affect the communications of the network. The next action of the attacker is the replay attack. This action is similar to eavesdropping as it requires the attacker to capture the information being shared between wireless nodes. After intercepting the information the replay attack stores, then resends the communications to other nodes to disrupt the network. Another purpose of the replay attack is to imitate a good node and attempt to gain access into the network by replaying authentication information. The attacker can also perform a Byzantine attack. This attack requires the attacker to intercept communications and modify them before forwarding the information to the recipient. This kind of attack can really corrupt a network that relies on proper communication protocols. Another action of the attacker is to perform a denial of service attack (DoS) where the attacker tries to overwhelm wireless nodes with communication in order to get the nodes to ignore legitimate traffic or even shut down entirely. This model also assumes that an attacker may not always be present and therefore adds that the attacker may do nothing as an action.

After the attacker's actions have been identified, the defender's actions that counteract the attacker's actions are found. This is where the simplicity of the model can be seen as the defender's actions are simply to defend against each action of

the attacker. This means, the defenders actions are defend eavesdropping, defend replay attack, defend byzantine attack, defend DoS, and defend nothing. For the purpose of analysis, these actions do not need to be specified directly in the model as in how they actually defend the attacks. Instead, the specifics of these actions become more important when determining the payoffs achieved when the attacker's action goes against a particular defender's actions. It is here that specific strategies of the defender are examined to determine the costs of these actions. The costs for the attacker's actions are also determined. The status of the mission is calculated by using thresholds. If the damage is limited to a certain threshold the defender receives a bonus, while the attacker receives a penalty if the damage is not above a certain threshold. This is shown in Equations 5 and 6. If the thresholds are exceeded then the penalty and the bonus drop to 0.

$$missionPenalty = \begin{cases} penalty & damage \leq attackerThreshold \\ 0 & damage > attackerThreshold \end{cases} \tag{5}$$

$$missionBonus = \begin{cases} bonus & damage \leq defenderThreshhold \\ 0 & damage > defenderThreshhold \end{cases} \tag{6}$$

The utility for the defender is determined by Equation 8 and the utility for the attacker is determined by Equation 7.

$$utility = missionBonus - damage - cost \tag{7}$$

$$utility = damage - cost - missionPenalty \tag{8}$$

The cost of the actions for the attacker and defender can be expressed in a multitude of ways. One way is through using real world data and figures. The values for the *cost*, *damage*, *missionBonus* and *missionPenalty* also need to be normal-

ized so they can properly be operated on in the equation. This can be done through some kind of conversion. In order to keep this model simple the cost of an action was based on the complexity of the action. A more complex action had a higher cost and the values ranged between 1 and 5. The complexity ranking for attacker's action starting with the most complex is byzantine attack, replay attack, eavesdropping, DoS and nothing. This ordering is determined logically. A denial of service attack can be accomplished many ways but with a wireless network it can much more easily by simply jamming the network with noise [49]. An eavesdropping attack can be considered more complex in that it requires the attacker to have a receiver that can capture and store transmitted data. The replay attack is more complex than eavesdropping because the attacker must not only capture and store the transmitted data but then be able to send it out again to be received by another node. Finally the Byzantine attack is more complex than the replay attack in that it requires the attacker to be able to modify the packets that it captures before sending them out back out. The cost and complexity for the defender's actions mirrored that of the attackers action meaning defending a byzantine attack is considered as complex and costly as the byzantine attack itself. The damage that a specific attack does against a certain defense is determined by how well that defense would mitigate that attack. Since there are cases where a defense may be able to defend against certain aspects of a non-matching attack it would be able to reduce the effect of that attack to some degree. Tables 21 and 22 shows the costs of the player's actions. Table 23 shows the damage each attack does against a specific defense.

Table 21. Defense Cost

Action	Cost
No Defense	0
DoS Defense	2
Eavesdropping Defense	3
Replay Defense	4
Byzantine Defense	5

Table 22. Attack Cost

Action	Cost
No Attack	0
DoS Attack	2
Eavesdropping Attack	3
Replay Attack	4
Byzantine Attack	5

Table 23. CDMF Model Damage Caused By Specific Attack on Specific Defense

	Attack				
Damage	**Replay Attack**	**Eavesdropping**	**DoS**	**Byzantine**	**No Attack**
0	Replay Def.	Eavesdropping Def.	DoS Def.	Byzantine Def.	Eavesdropping \ DoS \ Replay \ Byzantine \ No Def.
2	Eavesdropping Def.	Replay Def.	Byzantine Def.	Replay Def.	
4	Byzantine Def.	Byzantine Def.	Replay Def.	DoS Def.	
6	DoS Def.	DoS Def.	Eavesdropping Def.	Eavesdropping Def.	
12	No Def.	No Def.	No Def	No Def.	

As stated earlier, the cost of an attack is a value ranging from 1-5 that reflects how complex the attack is to accomplish. The amount of damage an attack causes against a particular defense is determined by how well the defense mitigates different aspects of the attack. For example, the replay attack costs 4 which makes it the second most complex option for the attacker. This also means the cost of a replay attack defense for the defender is also 4. The replay attack does 0 damage against a replay defense because it a perfect match. The replay attack does only 2 damage against an eavesdropping defense because this defense may mitigate some aspects of the attack. Meanwhile the replay attack does 6 damage against the DoS defense because this defense does not match up well with the attack. All attacks cause the maximum of

12 damage against a no defense action from the defender. For simplicity these values are meant to represent a ranking as opposed to actual damage that an attack would do against a particular defense in a real life scenario. The values for the mission status also need to be set. For this model the $attackerThreshhold$ was set to 1 and the penalty was set to 3. This means that, the attacker has to do more than 1 damage or they receive a penalty of 3 for failing the mission. The $defenderThreshhold$ was set at 2 and the bonus was set at 5. This means, that the defender receives a bonus of 5 if they are able to keep the damage at 2 or below. Using these values, the normal form game can be developed using the utility equations for the attacker and defender. The final result is the normal form game in Table 24.

Table 24. WSN CPS Defender vs. Attacker Normal Form

Defender\Attacker	Replay Attack	Eavesdropping	DoS	Byzantine Attack	No Attack
Replay Def.	1,-7	-1,-4	-8,-1	-1,-6	1,-3
Eavesdropping Def.	0,-5	2,-6	-9,1	-9,-2	2,-3
DoS Def.	-8,-1	-8,0	3,-5	-6,-4	3,-3
Byzantine Def.	-9,-3	-9,-2	-2,-3	0,-8	0,-3
No Defense	-12,5	-12,6	-12,7	-12,4	5,-3

These values are all based on ranking the actions and outcomes against each other rather than real world monetary costs or damage. This is done to make the model easier to understand. These values can also be easily modified as seen in Appendix B. The normal form game in Table 24 can be analyzed via HAT to determine the appropriate mixed strategy for the attacker and the defender to achieve a Nash equilibrium where each actor gains no incentive from deviating from their strategy. Due to the nature of the values used, the results need to be interpreted in a way to apply them to the real world. This is explored in Chapter IV. A hypergame analysis also

determines how the opponent can be outguessed to achieve a better outcome for each player which further develops the strategy moving forward.

This framework can be compared to the model developed by House and Cybenko [10] in that both were applied to a hypergame and feature attack specific defenses. However in the CDMF model the attacks are named specifically for the cyber attack of a WSN network of a CPS while the House and Cybenko model referred to the attacks generically as "attack x", "attack y", and "attack z". The purpose of that model was to prove a concept and not necessarily to apply to a specific situation as the CDMF model does. Also the CDMF model captures the idea of rewarding the defender for protecting the network and penalizing the attacker if they do nothing at all. This more accurately reflects real world payoffs for both the attacker and the defender that is not captured in the other model.

3.2 Method of Analysis

The proposed model is placed into a hyper normal form (HNF) hypergame. Hypergame analysis of this scenario allows us to see what different strategies can be taken based off different perceptions of the full game for the defender and even the attacker. The goal is to estimate a more realistic outcome for the defender because in the real world the defender is only assuming the possible attack vectors of the attacker and equally an attacker may only know so much about the environment they are trying to affect. This uncertainty can be captured in the hypergame to provide much more applicable analysis. The actions in the CDMF bare resemblance to the 4 canonical information warfare strategies that were also investigated with the earlier hypergame modeling technique [50] found in Fraser's research [51] . This technique however did not apply numerical analysis or facilitate the reasoning of strategies based around an equilibrium. HNF is chosen to model the scenario because it is simple to

understand as an extension of the standard normal form game. It allows multiple states of perception to be modeled all at one time, and can also utilize real world data to form its belief context percentages in order to gain a better solution. Another aspect of the HNF that is beneficial is that it can measure the belief in being outguessed. This allows a decision maker to gauge their expected payoff based on how confident they are in their beliefs. That means that a decision maker can instantly see the consequences of playing safe vs taking a risk. This is something that is only calculated in the HNF of hypergames as developed by Vane in his research.

3.3 HAT Tool

The HAT software is used to conduct analysis of the hypergames. The HAT software is capable of modeling two player hypergames in their normal form and find the hyper Nash equilibrium as well as the expected utility (EU) from all four of the hyperstrategies. First the user must enter in the parameters of the game which includes each player's strategies and the utilities they receive when these strategies are played against each other. One thing to note is that the HAT software has the additional feature of taking functional utilities as input which is absent in Gambit. This means, that the utilities received can be represented as functions of variables. Once the value of the variables are determined the utility value is calculated. This allows for a much simpler modification of games whose utilities are not static across all conditions. After inputing the strategies and the utilities for each player, the different game perceptions and pre-conceived game strategies for the column player are entered. Game perceptions represent the different ways the row player sees the column player perceiving the game and are referred to as subgames in HAT. For example in a hypergame of Rock Paper Scissors, the row player may believe the column player perceives a game where scissors is not an available strategy and only considers rock or paper as a strategy.

The HAT software finds the new Nash Equilibrium under these conditions which may alter the preferred strategy for the row player. Additionally row player can have a preconception of the probability distribution of the column player's strategies when they play the full game. The preconceptions for these probability distributions can be derived from research or observation. For example considering Rock, Paper Scissors again, the row player can have a preconception that the column player abandons the Nash equilibrium strategy and play Rock 20% of the time, Paper 10% of the time, and Scissors 70% of the time based off statistics of prior games. The difference in this pre-conceived game is that the player has an explicit belief that is not based off any sort of equilibrium calculation. Each of these perceived games and strategy preconceptions must be entered with a belief context which is a percentage representing how much the row player believes the column player is actually playing according to that game. The sum of the percentages for the belief contexts sum to 100%. Since the full game is displayed by default in HAT its belief context is calculated as the remaining belief context that has not been committed to other mixed strategies. Once this has all been entered HAT runs the hypergame and displays the EU, the worst case EU and the hyperstrategy EU (HEU) for all four of the different hyper strategies. The HEU can also be adjusted based on the "fear of being outguessed" variable which takes a value between 0 and 1. All these features make HAT a very useful piece of software for finding hyperstrategies and their EUs. However there are some aspects that needed to be improved to achieve a more comprehensive software seen in its game theory counterparts such as Gambit. One of the key improvements that was made in this effort was to simplify the process of adding belief context probabilities making the software more user friendly, dynamic and reducing the chances of calculation errors. Another improvement that was implemented was the addition of the mixed integer programming (MIP) method of finding the Nash equilibrium to go along with support

enumeration and the Lemke-Howson method that were currently present. Along with reducing the gap between the number of Nash equilibrium methods found in HAT and the more fleshed out Gambit software, the addition of the MIP method reliably produces multiple Nash equilibriums for a game if available and can be modified to find a user defined most optimal Nash equilibrium.

3.4 Update Belief Contexts

The HAT software is able to calculate hypergame Nash equilibriums based off the belief context percentages of column mixed strategies (CMS). In order for hypergame Nash equilibriums to be calculated properly the belief context percentages need to sum up to 1. HAT required the user to enter the context beliefs for each for each CMS manually including the full game CMS. This meant that it was up to the user to ensure that the CMS belief contexts added up to 1. The system was not set up to prevent context beliefs from summing less than or more than 1. Instead the HAT software would continue to calculate the hypergame Nash equilibriums incorrectly. In Figure 7 it is seen what would happen if a user did not enter belief contexts that added up to 1.

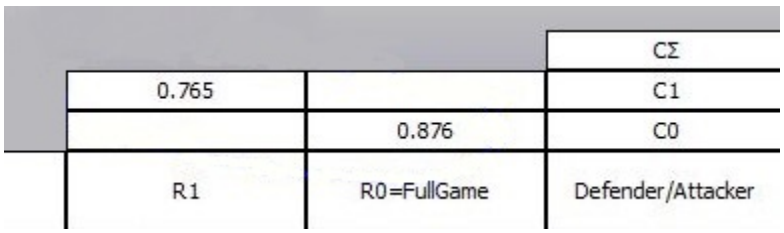

Figure 7. Belief Context Do Not Sum to 1.

This image shows the belief contexts for the two CMS of the game. The belief context for $C1$ is 0.765 which means that the row player believes the column player is playing this strategy with 76.5% certainty. The belief context for $C0$ is 0.876 which represents the row players 87.6% certainty that the column player is playing that

strategy. Adding the row player's beliefs gets $76.5\% + 87.6\% = 164.1\%$. Since it is impossible to have over 100% certainty in something it is no surprise that this leads to improper calculations of the HEUs. In order to make the HAT software more robust and help avoid incorrect calculations the $C0$ value calculation was changed so that it would no longer be entered manually, but instead be calculated dynamically based on the amount belief context that was distributed to other CMSs. If there is any belief context left after distributing amongst the other CMSs it would go to $C0$. Referring to Figure 8 it can be seen that the $CS1$ CMS has been allocated 0.765 belief context. Since the belief contexts must sum to 1 this means, there is $1 - 0.765 = 0.235$ belief context left to be assigned to other CMSs. $C0$ is the only other CMS and it gets the remaining uncommitted belief context of 0.235. This is calculated and entered automatically without the user having to enter 0.235 for $C0$.

		CΣ
0.765		CS1
	0.235	C0
RS1	R0=FullGame	Defender/Attacker

Figure 8. Belief Context Sums to 1.

This simplifies the process of entering the CMS belief contexts for the game while removing chances for human error making the system more robust. Another improvement that was added in this category was making the belief contexts for subgames game specific just like the belief contexts for CMSs with pre-conceived strategies. Prior to this update, HAT loaded with the same subgame belief context value being used across every game. If the user wanted to use a different value for a different game the software had to be closed, modified and loaded again. By adding this update HAT can be loaded with each game having their own unique belief context percentage for their subgame which makes HAT much more dynamic and user friendly.

3.5 Update Nash Equilibrium Methods

Another update to the HAT software is the addition of the MIP algorithm to calculate Nash equilibriums. This method was implemented into the HAT software using the IBM ILOG CPLEX Optimizer library version 12.7 [52] as recommended in the original MIP paper [31]. The code for this algorithm was completed in the Netbeans IDE version 8.1 and can be seen in Appendix C. Prior to this update HAT only found the Nash equilibrium through support enumeration and the Lemke-Howson method. Different methods to find a Nash equilibrium each have their own benefits. Some are easier to implement, some find multiple strategies, and some are able to find certain strategies faster. Since each method has their own benefit, having a variety allows all the different benefits to be explored. The MIP algorithm has been shown to be faster than the Lemke-Howson algorithm although it is slower than the support enumeration on average. For large games however, the MIP can be a good alternative to support enumeration and the Lemke-Howson method where they might have a hard time finding a solution quickly. Another benefit of the MIP algorithm is that it can find multiple Nash equilibriums in one run. This paper hypothesizes that different Nash equilibriums for a game can affect the calculations and final outcome of the hypergame strategies. Therefore the implementation of MIP into HAT allows the hypothesis to be tested. Another addition that MIP brings to the HAT software is the ability to find optimal Nash equilibriums based on a user's preferences. This makes it easier to find the Nash equilibrium strategy that best fits a particular circumstance in cases where there are multiple mixed strategies. For example there may be a Nash equilibrium strategy that has a smaller support for the column player when compared to another mixed strategy. The strategy with a smaller support for the column player may be more desirable for the row player because they have less actions from the

column player to worry about and prepare for. With this information the row player can plan to push for their desired equilibrium strategy in the game.

3.6 Chapter Summary

This chapter introduces the CDMF which is a simple and flexible approach to quickly model a cyber defense scenario. The CDMF is utilized to build a WSN CPS model that is under threat of an attacker. The benefit of this model is that it can be easily applied to a normal form or HNF game to conduct analysis. The HAT software makes it possible to conduct HNF analysis of this new model. The improvements to the software ensure its accuracy in calculations and allow multiple Nash equilibriums to be analyzed much more easily. In the following chapter the model developed here is analyzed using the improved HAT software to explore new analysis strategies for a hypergame.

IV. Design, Experiments and Results

4.1 Introduction

This chapter discusses the experiments that are run on a Blotto game and the new cyber-physical system (CPS) model created in Chapter III. First the updated Hypergame Analysis Tool (HAT) software is validated against classical games to ensure accurate results when running the experiment. Then HAT's application to games within the cyber arena is demonstrated by running it against cyber related models. Once this is established the experiments ran against the Blotto game and the new CPS model are explained, and the results are displayed and analyzed.

4.2 Updated HAT Validation

This section tests the mixed integer programming (MIP) algorithm in the updated HAT software against classic games to ensure its accuracy prior to running experiments on the Blotto game and new CPS model. The classic games were selected because they all have well established results. These tests ensure that the MIP algorithm added to HAT can solve games with a single pure strategy and a single mixed strategies. The test also validates the additional benefit of the MIP algorithm to find multiple mixed strategies if they are available and display them in the HAT software. The correctness of the results are all validated against results in GAMBIT and cross-checked with prior documented and peer reviewed results.

Test Single Pure Strategy.

This section tests HAT's ability to solve games with one pure strategy equilibrium. The support for each player only contains one strategy. This means, that if each player played there pure strategy they would have no reason to deviate from this strategy

given the other player's strategy. This is selected as the first test because dominant strategies are relatively easy to calculate compared to mixed strategies.

Prisoner's Dilemma, the most popular games that fits this category, is used to test this out. This game consists of two criminals who must determine whether they want to betray their partner for a chance at a reduced sentence or if they want to remain quiet and risk a maximum sentence. The expected results of this game is that both players choose to betray each other and receive a non-optimal solution. Figure 9 shows the Prisoner's Dilemma game entered into the HAT software and the calculated equilibrium. Although the utilities in this game are different than the version in Chapter II, the strategy is still the same. A value of '1' is seen next to the "Confess" strategy for the row player and the same for the column player. This means, that both players are playing their dominant strategy and are not mixing for the equilibrium. This aligns with the expected results and shows MIP was able to correctly identify the NE strategy for this kind of game.

	CΣ	0.0	1.0
1.0	C0	0.0	1.0
R0=FullGame	Prisoner A/Prisoner B	Stay Quiet	Confess
0.0	Stay Quiet	-1.00, -1.00	-5.00, 0.00
1.0	Confess	0.00, -5.00	-3.00, -3.00

Figure 9. Prisoner's Dilemma

Another popular game is the Deadlock game. It is very similar to the Prisoner's dilemma except that the payoffs in the game encourage the player's to cooperate instead of work against each other which produces a Pareto optimal outcome for the players [53]. Just like the Prisoner's Dilemma game, this game has only one pure strategy. Figure10 shows the results of the game using MIP. The values of '1' next to the players' strategies indicate the NE strategy is indeed a pure strategy for both players. Also the payoffs in the game show that the player's do achieve a Pareto

optimal solution as opposed to Prisoner's Dilemma game. This signifies that MIP algorithm is able to properly identify the NE in games with one pure strategy.

	CΣ	0.0	1.0
1.0	C0	0.0	1.0
R0=FullGame	Player1/Player2	B1	B2B
0.0	A1	1.00, 1.00	0.00, 3.00
1.0	A2	3.00, 0.00	2.00, 2.00

Figure 10. Deadlock

Test Single Mixed Strategy.

This section looks at MIP's ability to find mixed strategies in games that do not have pure strategies. These games do not have a pure strategy because there is always incentive for a player to change their strategy given the other player's strategy remains the same. Instead, the NE strategy is to mix their strategies among the available actions. This guarantees that the support of these strategies is greater than 1.

A popular game that fits this description is the classic Rock, Paper, Scissors game. In this game one player or the other can achieve a better payoff by switching their action given that the other player maintains there strategy. Therefore the NE strategy for each player is to randomize evenly over all three strategies. Figure 11 shows the value of '0.333' next to each player's actions. This means, that the NE for each player is to randomly play each of the three strategies evenly. This aligns with the expected result and shows that MIP was able to find the mixed strategy NE.

	CΣ	0.3333	0.3333	0.3333
1.0	C0	0.3333	0.3333	0.3333
R0=FullGame	Player1/Player2	Rock	Paper	Scissors
0.3333	Rock	0.00, 0.00	-1.00, 1.00	1.00, -1.00
0.3333	Paper	1.00, -1.00	0.00, 0.00	-1.00, 1.00
0.3333	Scissors	-1.00, 1.00	1.00, -1.00	0.00, 0.00

Figure 11. Rock Paper Scissors

The next game to consider is the Matching Pennies game. This is a two player zero sum game where each player simultaneously selects a face of a penny. One player's goal is to select the same face as the other player's selection, which is considered a match, while the other player's goal is to select the opposite face of the other player's selection. It can be seen how in this zero sum game where there are no ties, if each player knows what side the other selected there is one player who wants to switch their side given the other player keeps their selection. This means, that there is no pure strategy NE and only a mixed strategy NE. In Figure 12 the results of the game using MIP show that each player mixes evenly over the available strategies. This shows that MIP is able to solve games with a single mixed strategy NE.

	CΣ	0.5	0.5
1.0	C0	0.5	0.5
R0=FullGame	Player1/Player2	Heads	Tails
0.5	Heads	1.00, -1.00	-1.00, 1.00
0.5	Tails	-1.00, 1.00	1.00, -1.00

Figure 12. Matching Pennies

Test Multiple NEs.

In this section the MIP implementation in HAT is shown to be able to find multiple NE in games that have more than one mixed strategy. MIP finds an initial NE for a game, but continues to find more until it determines that no more strategies exist or the algorithm times out. All these strategies are stored and indexed. Different strategies can be easily accessed on individual runs in the HAT software simply by changing the index of the strategy MIP is referencing which is seen in Appendix C. This is an improvement over HATs original method of finding equilibriums strategies which only found 1 strategy per run. To test this feature MIP is used to run games known to have multiple NEs including pure and mixed strategies to see if all the strategies are found.

Battle of the Sexes is the first classic game with the desired property of multiple strategies that is tested against MIP in HAT. This is a two player non-zero sum game where a man and a woman are trying to decide what to do for an activity. Each player has their preference but they would rather agree to go to the same place then each go do separate activities. This means, that there is a desire for each player to cooperate and it can be seen logically that there are 2 pure strategies where both the man and women agree on the man's preferred activity or the woman's preferred activity and would not want to switch and be on their own. Less obvious is the mixed strategy where each player mixes over their options. In Figures 13, 14, and 15 the results from the game being analyzed in HAT show that all three strategies were found using the MIP algorithm.

	CΣ	1.0	0.0
1.0	C0	1.0	0.0
R0=FullGame	Male/Female	Bach	Stravinsky
1.0	Bach	3.00, 2.00	1.00, 1.00
0.0	Stravinsky	0.00, 0.00	2.00, 3.00

Figure 13. Battle of the Sexes: Pure Strategy 1

	CΣ	0.0	1.0
1.0	C0	0.0	1.0
R0=FullGame	Male/Female	Bach	Stravinsky
0.0	Bach	3.00, 2.00	1.00, 1.00
1.0	Stravinsky	0.00, 0.00	2.00, 3.00

Figure 14. Battle of the Sexes: Pure Strategy 2

	CΣ	0.25	0.75
1.0	C0	0.25	0.75
R0=FullGame	Male/Female	Bach	Stravinsky
0.75	Bach	3.00, 2.00	1.00, 1.00
0.25	Stravinsky	0.00, 0.00	2.00, 3.00

Figure 15. Battle of the Sexes: Mixed Strategy

The next classic game that is tested against the MIP algorithm is the Chicken game. This two player zero sum game looks at the situation where two cars are driving towards each other and one of the cars has to swerve to avoid a crash. Each driver would rather drive straight then swerve but would much rather swerve than crash. This game has two pure strategy NEs where one driver stays straight and the other swerves along with one mixed strategy where each driver mixes over their actions. The results in Figures 16, 17, and 18 show that all the strategies were found and MIP is capable of finding all the strategies in a game.

	CΣ	0.0	1.0
1.0	C0	0.0	1.0
R0=FullGame	Player1/Player2	Swerve	Straight
1.0	Swerve	0.00, 0.00	-1.00, 1.00
0.0	Straight	1.00, -1.00	-10.00, -10.00

Figure 16. Chicken: Pure Strategy 1

	CΣ	1.0	0.0
1.0	C0	1.0	0.0
R0=FullGame	Player1/Player2	Swerve	Straight
0.0	Swerve	0.00, 0.00	-1.00, 1.00
1.0	Straight	1.00, -1.00	-10.00, -10.00

Figure 17. Chicken: Pure Strategy 2

	CΣ	0.9	0.1
1.0	C0	0.9	0.1
R0=FullGame	Player1/Player2	Swerve	Straight
0.9	Swerve	0.00, 0.00	-1.00, 1.00
0.1	Straight	1.00, -1.00	-10.00, -10.00

Figure 18. Chicken: Mixed Strategy

4.3 Cyber Application

HAT is capable of analyzing games from a hyper normal form game standpoint. This proves useful in its ability to identify strategies that take into account knowledge

a player may have on the other. This kind of hypergame analysis has been found useful in the cyber arena when looking at attacker-defender models. This section looks at how HAT can analyze models that were originated in standard game theory to find new information and develop deeper strategies.

An intrusion detection model previously ran in game theory normal form [41] was placed into the HAT software to analyze how a defender would use knowledge gained on the attacker from playing sequential games to alter their beliefs in order to achieve a higher expected utility (EU) [9]. The results showed that a defender was able to learn more about what kind of attacker they were facing after every game. This information was used to improve their utility outcomes by updating their beliefs in the subgames after every round. Figure 19 shows a defender going against an attacker who they believe may be a nuisance attacker with a belief context of 0.6. Initially the hyperstrategies in the game are all ineffective as their EUs cannot do better than the Nash equilibrium mixed strategy (NEMS). As the game progressed their belief context in the attacker changed along with other payoffs variables to reflect what was learned about the attacker. This led to a better payoff compared to games where these updates were not made. This showed how HAT allowed an adaptive strategy for a defender to be modeled which could then be applied in persistent cyber defense planning. One thing to point out in this model is that the accomplishment of the mission is not accounted for in the case of the attacker as they receive no penalty for not attacking. One way to account for this is to get rid of the "no attack" option for the attacker but that would change the model of the game and limit the application of the results.

					CΣ	0.7481	0.0519	0.2
		0.2			CS1	0.9351	0.0649	NaN
			0.6		Nuissance - C1	0.9351	0.0649	0.0
				0.2	C0	0.0	0.0	1.0
PSS1	MO	RS1	R1	R0=FullGame	Defender/Attacker	Not Attack	Attack	Zero-Day Exploit
0.5	1.0	0.5	1.0	1.0	Not Defend	0.00, 0.00	-1.00, 0.90	-1.00, 1.00
0.5	0.0	0.5	0.0	0.0	Defend	-0.12, 0.00	0.70, -0.90	-1.00, 1.00
0.0	0.0	0.0	0.0	0.0	Provide Ruse	-2.00, 0.00	-1.00, 0.90	-1.00, 1.00
0.0	0.0	0.0	0.0	0.0	Shut Down	-3.00, 0.00	-3.00, -1.00	-2.00, -1.00
-0.2519	-0.2519	-0.2519	-0.2519	-0.2519	<-- **EU**			
-1.0	-1.0	-1.0	-1.0	-0.2519	<-- **G**			
-0.4015	-0.4015	-0.4015	-0.4015	-0.2519	<-- **HEG**			

Figure 19. Intrusion Detection Game in HAT

Next HAT is used on a CPS SCADA defense model originally modeled in a sequential game using the extensive game form [54]. In this game the attacker has 4 attacks at their disposal which are $a_s = sybil$, $a_{NC} = node\,compromise$, $a_e = eavesdropping$, $a_{DI} = data\,injection$, and $a_0 = do\,nothing$. The defender has two options which are defend by cutting off energy, alerting the MTU, and maintenance of the nodes or $< r_e, r_a, r_m >$ and the other option is $r_0 = do\,nothing$. In this game the attacker makes the first move, followed by the defender, and followed by the attacker again to complete the sequence. As shown in Figure 20 initially the attacker only has r_s and r_{NC} available and depending on the action of the defender in the next stage, the attacker has the options a_{DI}, a_e or a_0. In the extensive form of the game the defender can use backwards induction to find the equilibrium strategy sets which are (a_s, a_e, a_e, a_e) for the attacker and $(< r_e, r_a, r_m >, < r_e, r_a, r_m >)$ for the defender. The attacker's equilibrium strategy set is interpreted as what the attacker does at each of the decision nodes. At the first decision node the attacker chooses a_s and at the second decision node , starting from left to right on the bottom of the extensive tree in Figure 20, the attacker performs a_e with a_e being chosen for the attacker's third and fourth decision nodes as well. The defender's strategy set means they defend at both decision nodes. This boils down to the attacker performs a sybil attack followed by the defender defending against it. In a final response to this the attacker performs an eavesdropping attack. Although a system administrator may be able to

make plans based on these results, this extensive form is not capable of capturing different perceptions the defender may have of how the attacker might actually play the game. To accomplish this the game was placed in HAT as seen in Figure 21. In this 4×16 game the NEMS of the full game is the same as that in the extensive form game which means HAT was able to correctly identify the equilibrium. The hypergame also has a subgame where the defender perceives the attacker does not use the action sets (a_s, a_e, a_e, a_e),(a_s, a_e, a_e, a_{DI}),(a_s, a_e, a_{DI}, a_e), and $(a_s, a_e, a_{DI}, a_{DI})$. This means, that the defender believes the attacker does not try to eavesdrop after its sybil attack is met with defense. This may be based off prior knowledge that the attacker may not know how to eavesdrop once their sybil attack is compromised. The subgame equilibrium shows that the attacker still proceeds with a sybil attack and that the defender should defend the attack and not expect a follow up attack. The summary column mixed strategy (CMS) shows the attacker mixes a sybil attack followed up by eavesdropping strategy at 23.5% with the strategy of doing nothing after the sybil attack with 76.5% based on their prior beliefs. This can relieve some stress from the defender as they know the attacker probably does not resort to an initial node compromise even if they know they cannot eavesdrop after being defended. This shows how HAT can incorporate the perceptions of the defender to gain additional information into the scenario. The issue with this model is that it basically states the defender should defend against attacks which may be seen as an obvious to a planner.

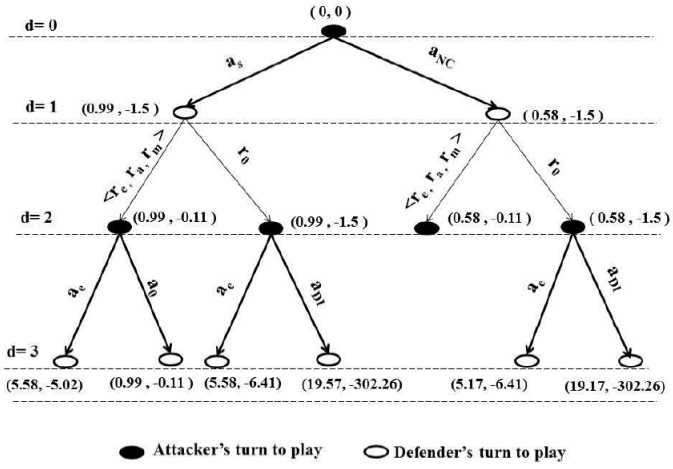

Figure 20. Sequential SCADA Game

					CΣ	0.235	0.0	0.0	0.0	0.0	0.0
			0.765		CS1	NaN	NaN	NaN	NaN	0.0	0.0
				0.235	C0	1.0	0.0	0.0	0.0	0.0	0.0
WS	PSS1	MO	RS1	R0=FullGame	Defender/Attacker	S, E, E, E	S, E, E, D	S, E, D, E	S, E, D, D	S, N, E, E	S, N, E, D
0.0	0.0	0.0	0.0	0.0	Defend, Not Defend	-5.02, 5.58	-5.02, 5.58	-5.02, 5.58	-5.02, 5.58	-0.11, 0.99	-0.11, 0.99
1.0	1.0	1.0	1.0	1.0	Defend, Defend	-5.02, 5.58	-5.02, 5.58	-5.02, 5.58	-5.02, 5.58	-0.11, 0.99	-0.11, 0.99
0.0	0.0	0.0	0.0	0.0	Not Defend, Defend	-6.41, 5.58	-6.41, 5.58	-302.26, 19.58	-302.26, 19.58	-6.41, 5.58	-6.41, 5.58
0.0	0.0	0.0	0.0	0.0	Not Defend, Not D...	-6.41, 5.58	-6.41, 5.58	-302.26, 19.58	-302.26, 19.58	-6.41, 5.58	-6.41, 5.58
-1.2646	-1.2646	-1.2646	-1.2646	-1.2646	<-- EU						
-5.023	-5.023	-5.023	-5.023	-1.2646	<-- G						
-2.0162	-2.0162	-2.0162	-2.0162	-1.2646	<-- HEG						

0.0	0.765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S, N, D, E	S, N, D, D	C, E, E, E	C, E, E, D	C, E, D, E	C, E, D, D	C, N, E, E	C, N, E, D	C, N, D, E	C, N, D, D
-0.11, 0.99	-0.11, 0.99	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18
-0.11, 0.99	-0.11, 0.99	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59
-302.26, 19.58	-302.26, 19.58	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59	-0.11, 0.59
-302.26, 19.58	-302.26, 19.58	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18	-6.41, 5.18	-302.26, 19.18

Figure 21. SCADA Game in HAT

The insider game [46] originally modeled in hypergames using HYPANT also benefits from HAT analysis. As described earlier this is a two player game between a system administrator trying to protect a network and an employee acting as an insider that is trying to exfiltrate data. The available actions to the attacker are probe for information or do nothing. The defender is capable of incident response or doing nothing as well. Hypergames are incorporated into the model by adding more actions into the model that the other player may not be aware of. The first added action is the strategem which allows the system administrator to trick the insider into probing false information and exposing themselves to being caught. The paper then

looks at the hypergame where the insider suspects that the system administrator is going to use a strategem and then alter their own strategy accordingly by choosing to do nothing. The last hypergame discussed is the game where the defender employs a fake or false strategem that uses much less resources than the strategem. The insider sees the false strategem and chooses to do nothing because they are unaware the defender has that option. The hypergames proposed did not use utility values, but instead employed preferred outcome vectors to rank the possible results achieved for both the defender and the insider. In order to convert this hypergame model into a hypergame normal form (HNF) model suitable for HAT, utility values were assigned equal to the ranking of the strategies in reverse order where a lower ranking is the worst strategy. Table 25 shows the outcome rankings for the insider and the defender. Note that the insider had outcomes that tied in ranking due to the source material not clearly distinguishing which outcome was more preferred. This is due to the false strategem not being available for the insider to consider in all of its hypergames. These rankings were used as utilities and placed in a hypergame as seen in Figure 22. In the full game it is seen that the equilibrium strategy for the attacker is a mixed strategy with "Probe" at 32.5% and do "Nothing" at 62.5%. The equilibrium strategy for the defender is to mix between using a strategem and false strategem evenly. This is interesting because it shows that the defender should not rely on incident response but rather deceptive tactics. This kind of numerical information was not able to be captured by HYPANT alone. This hypergame also contains a subgame where the defender has a 50% belief context that doing nothing is not an option for the insider. In this case the best strategy is to use a strategem which is the same strategy equilibrium strategy found in the original paper for the 1^{st} hypergame. This shows that HAT is capable of capturing the same information as HYPANT and more.

Table 25. Defender and Insider Outcome Rankings

Defender

Rank	Defender	Insider
8	False Strategem	No Action
7	Nothing	No Action
6	Strategem	Probe
5	Strategem	No Action
4	Incident Response	No Action
3	Incident Response	Probe
2	Nothing	Probe
1	False Strategem	Probe

Insider

Rank	Defender	Insider
6	Nothing	Probe
6	False Strategem	Probe
5	Incident Response	Nothing
4	Strategem	No Action
3	False Strategem	No Action
3	Nothing	No Action
2	Incident Response	Probe
1	Strategem	Probe

					CΣ	0.6875	0.3125
			0.5		CS1	1.0	NaN
				0.5	C0	0.375	0.625
WS	PSS1	MO	RS1	R0=FullGame	System Administrator/Insider	Probe	Nothing
0.0	0.0	0.0	0.0	0.0	Incident Response	3.00, 2.00	4.00, 5.00
0.75	1.0	1.0	1.0	0.5	Stratagem	6.00, 1.00	5.00, 4.00
0.25	0.0	0.0	0.0	0.5	False Stratagem	1.00, 6.00	8.00, 3.00
0.0	0.0	0.0	0.0	0.0	Nothing	2.00, 6.00	7.00, 3.00
5.0625	5.6875	5.6875	5.6875	4.4375	<-- EU		
4.75	5.0	5.0	5.0	4.4375	<-- G		
5.0	5.55	5.55	5.55	4.4375	<-- HEG		

Figure 22. Insider Game in HAT

4.4 Experimentation

The previous section showed the HAT software calculating accurate results while running MIP algorithm and how HAT can be used to model cyber defense scenarios. This section focuses back on the Blotto game as CPS model and the wireless sensor network (WSN) CPS model developed in Chapter III using the cyber defense modeling framework (CDMF). The models are input into HAT and solved with different configurations. In the Blotto game each player has three resources that must be distributed across three fronts. The Blotto game is ran multiple times with one subgame that is kept constant. Also the belief context for this subgame is kept constant at 0.5. The only thing changing is the equilibrium strategy of the full game. This experiment looks to capture the effects that changing the full game equilibrium strategy has on the hyperstrategies. An experiment is ran on WSN CPS model that

consists of switching the row player. Each configuration is ran once with a particular subgame meant to capture similar information. This is done to see if the decision maker gains new insight from switching the row player. All the results are analyzed following the experiment to see if the switch shed light on better strategies for the network defender.

Blotto Game.

The Blotto game was mentioned earlier as a popular classic game. It is known for being simple to conceptualize but having complex strategies and being hard to solve for large instances. Its simple construction allows it to be applied to many scenarios including cyber defense. For example the Blotto game was used to study how scarce protective resources could be automatically allocated in a CPS to combat the asset allocations of an attacker [55]. Blotto games have been researched in regards to players reasoning about their opponent's reaction to their strategy and how they should react to that. The reasoning continues to how the opponent would react to the player's reaction. Every step of reasoning is referred to as a k-level [56]. This idea of planning on how to endlessly react to an opponent's reactions to the player's reactions is not to be confused with hypergame theory that reasons over an opponent's perceptions of the game. However these two concepts can be combined to look at effects of misconceptions in k-level reasoning, although that is not explored in this research. The Blotto game in HAT allows for additional information to be added about the perception of an attacker to possibly gain a strategic advantage. In Figure 23 the Blotto game has been added to HAT. This is a two player game with three fronts and three resources per player. Each number digit in an action represent how many resources are assigned to particular front. For example the strategy "003" means 0 resources are assigned to the first and second front while the third front

receives all three of the resources. The full game strategy in this instance of HAT is to mix evenly between the three strategies of "120","012", and "201" for both players. This hypergame also has a subgame where the column player is not aware of a third front. Using this knowledge the row player can develop a better strategy to improve their outcome. The resulting hyperstrategies can be seen in Table 26. One strategy to consider is the MO hypergame strategy because it is fully effective. This means, that the MO strategy can perform better than the Nash equilibrium mixed strategy (NEMS) full game strategy in the best case while it cannot perform worse than NEMS in the worst case. The PS and WS are only partially effective strategies because their EUs is are above that of the NEMS, but both strategies are at risk of being outguessed and getting utilities lower than that of the NEMS worst case.

					CE	0.4167	0.25	0.0	0.0	0.0	0.1666	0.0	0.0	0.1666	0.0
			0.5		CS1	0.5	0.5	0.0	0.0	NaN	0.0	NaN	0.0	NaN	0.0
				0.5	C0	0.3333	0.0	0.0	0.0	0.0	0.3333	0.0	0.0	0.3333	0.0
WS	PSS1	MO	RS1	R0=FullGame	Colonel A/Colonel B	120	210	300	030	021	012	003	111	201	102
0.1666	0.0	0.0	0.0	0.3333	120	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	210	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00
0.0	0.0	0.0	0.0	0.0	300	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	030	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	-1.00, 1.00
0.25	0.5	1.0	0.5	0.0	021	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00
0.1666	0.0	0.0	0.0	0.3333	012	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	003	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	111	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.4166	0.5	0.0	0.5	0.3333	201	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	102	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.125	0.25	0.25	0.25	0.0	<-- EU										
-0.25	-0.5	0.0	-0.5	0.0	<-- G										
0.05	0.1	0.2	0.1	0.0	<-- HEG										

Figure 23. Blotto Game Strategy# 1

Table 26. Hyperstrategies # 1

Hyperstrategy	EU	G
MO	0.25	0
PS	0.25	-0.5
WS	0.125	-0.25
NEMS	0	0

The next Blotto game in Figure 24 features a pure strategy for the full game where the each player plays the strategy "111". In this strategy each player places a resource at one of the fronts. In this game there are no fully effective hyperstrategies

as seen in Table 27. Instead all the hyperstrategies are only partially effective as they do worse that the NEMS in the worst case scenario. The MO strategy is still better than PS or WS hyperstrategy as they have worse case scenarios that are worse and no real benefit on their mixed strategies.

					CΣ	0.25	0.0	0.25	0.0	0.0	0.0	0.0	0.5	0.0	0.0
			0.5		CS1	0.5	0.0	0.5	0.0	NaN	0.0	NaN	0.0	NaN	0.0
				0.5	C0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
WS	PSS1	MO	RS1	R0=FullGame	Colonel A/Colonel B	120	210	300	030	021	012	003	111	201	102
0.0	0.0	0.0	0.0	0.0	120	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	210	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00
0.0	0.0	0.0	0.0	0.0	300	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	030	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	-1.00, 1.00
0.25	0.5	0.0	0.5	0.0	021	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00
0.0	0.0	0.0	0.0	0.0	012	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	003	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.5	0.0	1.0	0.0	1.0	111	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.25	0.5	0.0	0.5	0.0	201	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	102	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.25	0.25	0.25	0.25	0.25	<-- EU										
-0.25	-0.5	0.0	-0.5	0.25	<-- G										
0.15	0.1	0.2	0.1	0.25	<-- HEG										

Figure 24. Blotto Game Strategy# 2

Table 27. Hyperstrategies # 2

Hyperstrategy	EU	G
NEMS	0.25	0.25
MO	0.25	0.0
WS	0.25	-0.25
PS	0.25	-0.5

In the 3^{rd} Blotto game in Figure 25 the row player and the column player are playing different strategies. The row player plays the strategy "111" while the column player mixes evenly between the strategies "120", "012", and "201". Table 28 shows that in this game MO is not an effective hypergame strategy and the player may as well play the NEMS which happens to be the same strategy. In this game the PS strategy and the WS are both partially effective with PS strategy having a higher EU at the risk of being outguessed.

					CΣ	0.4167	0.25	0.0	0.0	0.0	0.1666	0.0	0.0	0.1666	0.0
			0.5		CS1	0.5	0.5	0.0	0.0	NaN	0.0	NaN	0.0	NaN	0.0
				0.5	C0	0.3333	0.0	0.0	0.0	0.0	0.3333	0.0	0.0	0.3333	0.0
WS	PSS1	MO	RS1	R0=FullGame	Colonel A/Colonel B	120	210	300	030	021	012	003	111	201	102
0.0	0.0	0.0	0.0	0.0	120	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	210	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00
0.0	0.0	0.0	0.0	0.0	300	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	030	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	-1.00, 1.00
0.25	0.5	0.0	0.5	0.0	021	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00
0.0	0.0	0.0	0.0	0.0	012	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	003	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.5	0.0	1.0	0.0	1.0	111	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.25	0.5	0.0	0.5	0.0	201	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	102	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.125	0.25	0.0	0.25	0.0	<-- EU										
-0.25	-0.5	0.0	-0.5	0.0	<-- G										
0.05	0.1	0.0	0.1	0.0	<-- HEG										

Figure 25. Blotto Game Strategy# 3

Table 28. Hyperstrategies # 3

Hyperstrategy	EU	G
PS	0.25	-0.5
WS	0.125	-0.25
NEMS	0.0	0.0
MO	0.0	0.0

The 4^{th} Blotto game in Figure 26 is similar to the 3^{rd} game with the row player playing the strategy "111" and the column player also mixes evenly between three actions again except the actions are different this time. The column player's strategy is evenly mixed between "210", "021", and "102". Table 29 shows how all the hyperstrategies are ineffective against the NEMS. MO is the next best hyperstrategy as it has the best EU and best worst case scenario compared to PS and WS. The WS is the next hyperstrategy to consider, and PS is the worst hyperstrategy out of them with its EU being equal to the WS's worst case scenario.

					CΣ	0.25	0.1666	0.25	0.0	0.1666	0.0	0.0	0.0	0.0	0.1666
			0.5		CS1	0.5	0.0	0.5	0.0	NaN	0.0	NaN	0.0	NaN	0.0
				0.5	C0	0.0	0.3333	0.0	0.0	0.3333	0.0	0.0	0.0	0.0	0.3333
WS	PSS1	MO	RS1	R0=FullGame	Colonel A/Colonel B	120	210	300	030	021	012	003	111	201	102
0.0	0.0	0.0	0.0	0.0	120	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	210	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00
0.0	0.0	0.0	0.0	0.0	300	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	030	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	-1.00, 1.00	-1.00, 1.00
0.25	0.5	0.0	0.5	0.0	021	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00
0.0	0.0	0.0	0.0	0.0	012	-1.00, 1.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	003	-1.00, 1.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00
0.5	0.0	1.0	0.0	1.0	111	0.00, 0.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.25	0.5	0.0	0.5	0.0	201	1.00, -1.00	0.00, 0.00	0.00, 0.00	1.00, -1.00	0.00, 0.00	-1.00, 1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.0	0.0	0.0	0.0	0.0	102	0.00, 0.00	-1.00, 1.00	0.00, 0.00	1.00, -1.00	1.00, -1.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00	0.00, 0.00
0.2083	0.1667	0.25	0.1667	0.25	<-- EU										
-0.25	-0.5	0.0	-0.5	0.25	<-- G										
0.1167	0.0333	0.2	0.0333	0.25	<-- HEG										

Figure 26. Blotto Game Strategy# 4

Table 29. Hyperstrategies # 4

Hyperstrategy	EU	G
NEMS	0.25	0.25
MO	0.25	0.0
WS	0.2083	0.1667
PS	0.1667	0.0333

These Blotto games all demonstrate how hypergames can be utilized in a cyber scenario where a defender is using deception to hide certain fronts from the attacker. This extra information allows the defender to put together a plan that takes these beliefs into account to formulate better results than those obtained from the NEMS. This case is most compelling in the Blotto game in 23 because the decision maker is presented with the MO option which is fully effective against the NEMS. Therefore this may be a strategy that the row player might want to consider since the row player has nothing to lose with the strategy and can only gain. A decision maker may want to push the column player towards playing with the select equilibrium mixed strategy in the first Blotto game in Figure 23.

WSN CPS Model.

The WSN CPS model developed in Chapter III modeled a network administrator protecting a wireless nodes from compromise by an attacker. Typically games are analyzed from the defender's perspective so the results of the game can instruct the defender how to act. This experiment first runs the game with the defender as the row player with a subgame where the attacker is aggressive and does not consider no action or eavesdropping attacks. The subgame has a belief context of 0.3 showing a slight belief that the attacker is aggressive. Figure 27 shows the results from that game. The game shows that the row player believes the attacker may try to use the eavesdrop, DoS, or replay attack. Table 30 shows the values of the hyperstrategies of this game. MO is partially effective and has the best EU but it is a very risky strategy leaving the defender open to eavesdropping and replay attacks and having a worst case utility of -8. In this case the defender may want to simply go with the NEMS as all other hyperstrategies are ineffective compared to it. The problem with this strategy is that it calls the defender to ignore the possibility of replay attacks that the defender believes the attacker may attempt in the subgame. PS also ignores the possibility of DoS attacks while also being ineffective. This leaves the WS. While this game is ineffective against the NEMS it takes into account the belief that the attacker may be playing a subgame and places some percentage in all three of the possible necessary defenses of eavesdrop, DoS, and replay defense. This calls into question the way these games can be interpreted in a cyber scenario. As stated in the "Security" section of Chapter II defenses in a security system are ran concurrently. This mean the defender employs defenses against all the attacks at once and does not randomize over a mixed strategy to achieve an equilibrium. This is because a network administrator cannot risk leaving any avenue open. So in this case the results of the hypergame cannot be applied as they are typically to other non-cyber defense

scenarios. In this case it is best to look at the hyperstrategies as a guide of emphasis on actions. For example looking at the WS strategy in Figure 27 it suggest that defender should expect a DoS attack the most, followed by eavesdropping, and to an even lesser extent replay attacks. Defenses assigned 0 probability, which in this case is the byzantine attack, should be the least expected. At the end of the day the defender protects against all potential attacks but based off the WS, which takes into account all subgames and belief contexts, the defender knows what degree to expect these attacks.

					CΣ	0.165	0.3819	0.4531	0.0	0.0
			0.3		CS1	0.55	NaN	0.45	0.0	NaN
				0.7	C0	0.0	0.5455	0.4545	0.0	0.0
WS	PSS1	MO	RS1	R0=FullGame	Defender/Attacker	Replay Attack	Eavesdropping	DOS	Byzantine Attack	No Attack
0.12	0.4	0.0	0.4	0.0	Replay Def.	1.00, -7.00	-1.00, -4.00	-8.00, -1.00	-1.00, -6.00	1.00, -3.00
0.2917	0.0	0.0	0.0	0.4167	Eavesdropping Def.	0.00, -5.00	2.00, -6.00	-9.00, 1.00	-9.00, -2.00	2.00, -3.00
0.5883	0.6	1.0	0.6	0.5833	DOS Def.	-8.00, -1.00	-8.00, 0.00	3.00, -5.00	-6.00, -4.00	3.00, -3.00
0.0	0.0	0.0	0.0	0.0	Byzantine Def.	-9.00, -3.00	-9.00, -2.00	-2.00, -3.00	0.00, -8.00	0.00, -3.00
0.0	0.0	0.0	0.0	0.0	No Defense	-12.00, 5.00	-12.00, 6.00	-12.00, 7.00	-12.00, 4.00	5.00, -3.00
-3.2019	-3.346	-3.0159	-3.346	-3.1401	<-- **EU**					
-6.2751	-5.2	-8.0	-5.2	-3.1401	<-- **G**					
-3.8165	-3.7168	-.81272	-3.7168	-3.1401	<-- **HEG**					

Figure 27. WSN CPS Model: Defender

Table 30. WSN CPS Model: Defender Hyperstrategies

Hyperstrategy	EU	G
MO	-3.0159	-8.0
NEMS	-3.1401	-3.1401
WS	-3.2019	-6.2751
PS	-3.346	-3.7168

The next game run of the WSN CPS model in HAT has the attacker as the row player. In the previous game in Figure 27 the attacker was suspected to be aggressive and not considering no action or eavesdropping in the subgame. This led the attacker to favor the replay and DoS attacks. The subgame in the attacker row player game checks to see if the attacker would still attempt these actions if the defender left these avenues open. Therefore the subgame has the defender not considering a replay

defense or a DoS defense. The belief context is still set at 0.3. The results for this game are seen in Figure 28 and the hyperstrategies are seen in Table 31. The MO is the only partially effective strategy and it does not even have the worst possible outcome. Attackers are different than defenders in that they sometimes choose to do one attack at a time or all at once because sometimes different attacks can interfere with each other. Therefore the interpretation of the mixed strategies as randomly mixing between actions works better with the attacker. The MO recommends that the attacker try a DoS attack although this strategy requires a strong belief g value of at least 0.89867 in the subgames. Anything less than that the attacker should mix their strategy between DoS attacks and eavesdropping as seen in the NEMS. An interesting thing to notice in this game is that if the defender opens themselves up to the replay attack and DoS attacks the attacker would rather abandon replay attacks all together and attempt Byzantine attacks with 0.875 compared to the 0.125 percentage for DoS attacks. This is interesting as the defender would expect the attacker to exploit the lack of a replay defense with a replay attack. This contrast the defenders CPS game where Byzantine attacks seemed to be the least of their worries even with an aggressive attacker. This information can be used by the defender to ensure that they do not slack on Byzantine defense. However since the Byzantine defense can be seen as expensive the defender can ensure the replay attack defense is solid which is cheaper as it seems that also deters the attacker from using a Byzantine attack.

					CΣ	0.0	0.3819	0.3181	0.1167	0.1833
			0.3		CS1	NaN	0.0	NaN	0.3889	0.6111
				0.7	C0	0.0	0.5455	0.4545	0.0	0.0
WS	PSS1	MO	RS1	R0=FullGame	Attacker/Defender	Replay Def.	Eavesdropping Def.	DOS Def.	Byzantine Def.	No Defense
0.0	0.0	0.0	0.0	0.0	Replay Attack	-7.00, 1.00	-5.00, 0.00	-1.00, -8.00	-3.00, -9.00	5.00, -12.00
0.2917	0.0	0.0	0.0	0.4167	Eavesdropping	-4.00, -1.00	-6.00, 2.00	0.00, -8.00	-2.00, -9.00	6.00, -12.00
0.4458	0.125	1.0	0.125	0.5833	DOS	-1.00, -8.00	1.00, -9.00	-5.00, 3.00	-3.00, -2.00	7.00, -12.00
0.2625	0.875	0.0	0.875	0.0	Byzantine Attack	-6.00, -1.00	-2.00, -9.00	-4.00, -6.00	-8.00, 0.00	4.00, -12.00
0.0	0.0	0.0	0.0	0.0	No Attack	-3.00, 1.00	-3.00, 2.00	-3.00, 3.00	-3.00, 0.00	-3.00, 5.00
-1.1254	-1.9912	-0.2756	-1.9912	-0.7543	**<-- EU**					
-4.0208	-7.375	-5.0	-7.375	-0.7543	**<-- G**					
-1.7045	-3.068	-0.77952	-3.068	-0.7543	**<-- HEG**					

Figure 28. WSN CPS Model: Defender

Table 31. WSN CPS Model: Attacker Hyperstrategies

Hyperstrategy	EU	G
MO	-0.2756	-5.0
NEMS	-0.7543	-0.7543
WS	-1.1254	-1.7045
PS	-1.9912	-3.068

4.5 Summary of Experiments

In this chapter HAT is validated against classical models and its application to cyber defense scenarios are shown. It is also shown in the experiments that when analyzing a hypergame in its normal form it is important to recognize the equilibrium strategy being used to make the calculations. This is because, when other factors are kept constant, changing equilibrium strategies in a hypergame also changes the viability of its hyperstrategies. These hyperstrategies may go from fully effective, to partially effective, to completely ineffective. This chapter also shows how looking at a game from the perspective of the attacker can provide additional insight for the defender to use in their planning. The following chapter discusses the conclusion of these findings.

V. Conclusion

The goal of this research is to further develop the field of hypergame theory, specifically as it pertains to cyber systems by developing an improved modeling and analysis environment for hypergame models. The goal of developing a new hypergame model and analyzing it with AFIT's Hypergame Analysis Tool (HAT) software's new Nash equilibrium method has been achieved by accomplishing the objectives laid out in Chapter I. This is done through a thorough study of game theory and hypergame theory literature and further development of the hypergame software package in relation to cyber-physical sytems (CPS). HAT is augmented to calculate Nash equilibriums (NE) using mixed integer programming (MIP). This implementation of MIP improves the processing of different equilibrium strategies to be displayed in HAT. HAT is validated against classical games and hypergames as well. HAT is also further improved by streamlining the task of calculating belief contexts in the hypergame making the overall software more robust and dynamic. More cyber-related games are added into HAT for analysis including an insider game, supervisory control and data acquisition (SCADA) game, and the Blotto game. This satisfies the first objective. On top of the established games a new model for a wireless sensor network (WSN) cyber-physical systems (CPS) is developed using the cyber defense modeling framework (CDMF). The CDMF model improves on other models in its simplicity while not being too generic. Another key benefit of CDMF is that it factors in the idea of mission success and failure into the utility calculations. This ensures that the attacker and the defender have incentives to achieve their goals and not avoid specific strategies due to fear of cost. The CDMF model's development completes the second objective.

The results from the experiments in Chapter IV show that changing different factors of the hypergame do in fact change the preferred hyperstrategies. By applying

these hyperstrategies a planner can hope to increase their expected utility (EU) over that of the Nash equilibrium mixed strategy (NEMS) at the risk of being outguessed. The results from the WSN CPS experiments show that changing the player can provide additional insight into developing a strategy for the defender. The experiments also called into question the traditional interpretation of mixed strategies in regards to cyber defense models. Since a defender is likely to always take every defensive action the mixed strategy should not be seen as telling the defender what actions to take. Instead it should let the defender know what defenses they can expect to use the most. This can help the allocation of resources and additional preparation while still defending against all other threats. The discovery of this information through unconventional analysis techniques accomplishes the task set out in the third objective.

5.1 Future Work

This paper explored the effects on hyperstrategies when the full game NE strategy was changed. However the compounded effects of changing the subgame NE strategies along with the full game NEMS still needs to be researched. This may allow decision makers to find strategies that are even more tailored to their specific circumstances and meet specific requirements. Also more research into a possible pattern in how the equilibrium changes affect the hypergame strategies would allow a planner to better anticipate analysis results.

The HAT software can use some future improvements to make it the premier hypergame modeling software. The HAT software would benefit from additional methods of calculating the NE. Different methods of calculating the NE can lead to speed improvements in the software as well as different strategies for the cyber defender. The addition of more NE methods furthers HAT software development

into a fully realized hypergame suite. Also the MIP implementation into HAT is capable of finding optimal strategies based off of user defined criteria. This feature can prove useful by placing constraints on the size of a support and researching its benefits.

Another future research consideration is the addition of a temporal framework into the HAT software. The benefits of this addition to the software have been documented to provide the ability to model timing into a hypergame and sequential play [57]. This addition can allow for even more dynamic and accurate models of cyber defense interactions unique to today's war-fighter.

5.2 Final Remarks

The insight and improved tools presented in this thesis research allows the war-fighter to quickly construct a realistic model of a specific cyber scenario and gain insight into how the attacker should be approached to achieve the best outcome. Given that there are often misconceptions in real world cyber scenarios these tools and methods can help the cyber defenders of the Air Force military and the world make more informed decisions, instead of having to ignore the additional information at hand. Overall this research has been able to show how new techniques in hypergame analysis are able to draw more information from the same game scenario and provide a strategic advancement for the decision maker.

Appendix A. Lemke-Howson Calculation

This appendix shows the calculation of the mixed strategy Nash equilibrium using the Lemke-Howson Method discussed in Chapter II.

For this example, consider the two player $m \times n$ game in Table 32.

The algorithm starts by generating two matrices for each player which are matrix A and B corresponding to player A and B's utilities respectively.

$$A = \begin{vmatrix} 2 & 4 & 0 \\ 0 & 0 & 3 \\ 3 & 2 & 2 \end{vmatrix} \qquad B = \begin{vmatrix} 3 & 2 & 0 \\ 2 & 4 & 2 \\ 0 & 0 & 4 \end{vmatrix}$$

The mixed strategies found by this algorithm will be in the form of a m sized vector, x, for player A and a n sized vector, y, for player B. According to the algorithm the constraints for Nash equilibrium solution are $A^i y \leq 1$ and $x^T B_j \leq 1$. Non-negative slack variables r and s are used to turn these inequalities into the equalities $Ay + r = 1$ and $B^T x + s = 1$. Now set up the two tableaux that solve for the slack variables as seen in tableau A and tableau B.

$$r_1 = 1 - 2y_4 - 4y_5 \tag{A1}$$

$$r_2 = 1 - 3y_6 \tag{A2}$$

$$r_3 = 1 - 3y_4 - 2y_5 - 2y_6 \tag{A3}$$

Table 32. Lemke-Howson Example Game

Player A \Player B	4	5	6
1	(2, 3)	(4,2)	(0, 0)
2	(0, 2)	(0,4)	(3,2)
3	(3,0)	(2, 0)	(2,4)

$$s_4 = 1 - 3x_1 - 2x_2 \tag{B1}$$

$$s_5 = 1 - 2x_1 - 4x_2 - 2x_3 \tag{B2}$$

$$s_6 = 1 - 4x_3 \tag{B3}$$

In this algorithm x and r are seen as pairs along with y and s and labels refer to the index of a variable solved for in an equality. For example the equality $r_1 = 1-2y_4-4y_5$ provides the label 1 from the index of r. The goal of the algorithm is to start with all the labels and then pivot the variables until all the labels are covered again by the union of the two tableaux. For this example the labels are 1 through 6 and as stated earlier the algorithm starts with all the labels. A pivot is performed by choosing a random index to solve for which will cause a label to be lost. The lost label becomes the new index to solve for and this is repeated until all the labels are accounted for. For example If the previous equality is solved for y_4 it becomes $y_4 = \frac{1}{2} - \frac{r_1}{2} - 2y_5$. In this equality, the label is now 4 and the label 1 is lost. When choosing which equality to solve an index for the general rule is to choose the equality with a minimum value for that index. If there is a tie than randomly pick one. To avoid unending loops if the tie is formed again break the tie differently. The first index to pivot can be random and this example solves for x_1 in tableau B. The minimum value rule means use equation $B1$. This creates the new tableau B‘.

$$x_1 = \frac{1}{3} - \frac{2}{3}x_2 - \frac{1}{3}s_4 \tag{B‘1}$$

$$s_5 = \frac{1}{3} - \frac{8}{3}x_2 - 2x_3 + \frac{2}{3}s_4 \tag{B‘2}$$

$$s_6 = 1 - 4x_3 \tag{B‘3}$$

After pivoting on x_1 and plugging in the variable for the other equations, label 4 is lost since there is no longer an equation solving for a variable with an index of 4 which used to be s_4. In order to get it back, pivot on y_4 in tableau A because y and s are pairs. The minimum value rule dictates that the $A3$ be used. This creates the new tableau A‘.

$$r_1 = \frac{1}{3} - \frac{10}{3}y_5 + \frac{2}{3}y_6 + \frac{1}{3}r_3 \tag{A‘1}$$

$$r_2 = 1 - 3y_6 \tag{A‘2}$$

$$y_4 = \frac{1}{3} - \frac{2}{3}y_5 - \frac{2}{3}y_6 - \frac{1}{3}r_3 \tag{A‘3}$$

With tableau A updated, label 3 is now lost . In order to get it back, pivot on x_3 in $B‘3$ to get tableau B“.

$$x_1 = \frac{1}{3} - \frac{2}{3}x_2 - \frac{1}{3}s_4 \tag{B“1}$$

$$s_5 = -\frac{1}{6} - \frac{8}{3}x_2 + \frac{2}{3}s_4 - \frac{1}{2}s_6 \tag{B“2}$$

$$x_3 = \frac{1}{4} - \frac{1}{4}s_6 \tag{B“3}$$

After updating tableau B label 6 is lost so the next pivot is y_6 in $A‘2$ to get tableau A“.

$$r_1 = \frac{5}{9} - \frac{10}{3}y_5 - \frac{2}{9}r_2 + \frac{1}{3}r_3 \tag{A“1}$$

$$y_6 = \frac{1}{3} - \frac{1}{3}r_2 \tag{A“2}$$

$$y_4 = \frac{1}{9} - \frac{2}{3}y_5 + \frac{2}{9}r_2 - \frac{1}{3}r_3 \tag{A“3}$$

Label 2 was lost so the next pivot is x_2 on B“2 to get tableau B‘‘‘.

$$x_1 = \frac{7}{24} - \frac{5}{12}s_4 + \frac{1}{8}s_6 + \frac{1}{4}s_5 \tag{B‘‘‘1}$$
$$x_2 = \frac{1}{16} + \frac{1}{8}s_4 - \frac{3}{16}s_6 - \frac{3}{8}s_5 \tag{B‘‘‘2}$$
$$x_3 = \frac{1}{4} - 1 - 4x_3 \tag{B‘‘‘3}$$

Now label 5 is gone, so it needs to be brought back by solving for y_5 in A“1 to get tableau A‘‘‘.

$$y_5 = \frac{1}{6} - \frac{1}{15}r_2 + \frac{1}{10}r_3 - \frac{3}{10}r_1 \tag{A‘‘‘1}$$
$$y_6 = \frac{1}{3} - \frac{1}{3}r_2 \tag{A‘‘‘2}$$
$$y_4 = \frac{12}{45}r_2 - \frac{6}{15}r_3 + \frac{1}{5}r_1 \tag{A‘‘‘3}$$

After that pivot all the labels are accounted for with the union of both tableau. Now the mixed equilibrium can be found by solving for the variables in the equation. This is done by assigning a 0 to all the slack variables. The result of this can be seen in equation 9.

$$
\begin{aligned}
x_1 &= \tfrac{7}{24} \\
x_2 &= \tfrac{1}{16} \\
x_3 &= \tfrac{1}{4} \\
y_5 &= \tfrac{1}{6} \\
y_6 &= \tfrac{1}{3} \\
y_4 &= 0
\end{aligned}
\tag{9}
$$

These values need to be normalized by dividing each value by the sum of its entire vector. The final values are $x_1 = 0.482$, $x_2 = 0.103$, $x_3 = 0.413$ for the x vector and $y_4 = 0$, $y_5 = 0.333$, $y_6 = 0.667$ for the y vector.

Appendix B. Wireless Sensor Network Cyber Physical System Game

```
package com.nicholaskovach.jhalf.hypergame.generators.games;

import com.nicholaskovach.jhalf.components.RandomIDUtilities;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.GameData;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.ObjectFactory;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.OpponentStrategy;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.Outcome;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.Player;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.PlayerStrategy;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.Variable;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.VariableUpdate;
import com.nicholaskovach.jhalf.hypergame.generators.GameGenerator;
import java.math.BigDecimal;

/**
*
* @author nskovach
*/
public class CyberPhysicalWirelessSensorGameGenerator extends GameGenerator {

  public CyberPhysicalWirelessSensorGameGenerator() {

  }

  public GameData generateGameDataObject() {

    // Get a random ID generator/tracker
    RandomIDUtilities rID = new RandomIDUtilities(RandomIDUtilities.BYTES8);

    // Factory required to create game:
    ObjectFactory factory = new ObjectFactory();

    // Create the game:
    GameData gameData = factory.createGameData();
    gameData.setDisplayName("CPS Wireless Sensor");
    gameData.setUID(rID.getRandomIDForSession());
    gameData.setBeliefInSubLevels(BigDecimal.valueOf(.3));

    boolean reverse = true;

    boolean hideReplayD = true;
    boolean hideEavesD = false;
    boolean hideDosD = true;
    boolean hideByzD = false;
    boolean hideNoD = false;

    boolean hideReplayA = false;
    boolean hideEavesA = false;
    boolean hideDosA = false;
```

```
    boolean hideByzA = false;
    boolean hideNoA = false;

    /*
    Damage caused by specfic attack on against specific defense

    DAMAGE TABLE
    |Replay Attack|Eavesdropping Attack|DOS|Byzantine Attack| No Attack|
    Replay Defense---------- | ------0------ |  ---------2---------- |-4-|  -------2-------- | -----0---- |
    Eavesdropping Defense---| ------2------ |  ---------0---------- |-6-|  -------6-------- | -----0---- |
    DOS Defense------------ | ------6------ |  ---------6---------- |-0-|  -------4-------- | -----0---- |
    Byzantine Defense------ | ------4------ |  ---------4---------- |-2-|  -------0-------- | -----0---- |
    No Defense------------- | -----12------ |  --------12---------- |12-|  -------12------- | -----0---- |
    */

    double [][] damage = new double[][]{{0,2,4,2,0},
      {2,0,6,6,0},
      {6,6,0,4,0},
      {4,4,2,0,0},
      {12,12,12,12,0}
    };
    /*Mission Accomplished/Failed

    If the attacker doesn't cause enough damage then he sees a penalty for not completing their mission.

    If the defender stops the attacker from getting a certain damage then they get a bonus

    */
    int damagemin = 2;
    int damagemax = 3;
    int bonus =5;
    int penalty = 3;

    int [][] missionAccomplished = new int [5][5];
    int [][] missionFailed = new int [5][5];

    for (int i = 0;i<5;i++){
      for(int k = 0;k<5;k++){
        if(damage[i][k] < damagemin ){
          missionFailed[i][k] = penalty;
        }else{
        missionFailed[i][k] = penalty;
      }
      if(damage[i][k] < damagemax ){
        missionAccomplished[i][k] = bonus;
      }else{
      missionAccomplished[i][k] = 0;
    }
  }
}


// Create the first player:
Player pA = factory.createPlayer();
```

```
pA.setUID(rID.getRandomIDForSession());
pA.setDisplayName("Defender");
pA.setRowPlayer(!reverse);
gameData.getPlayer().add(pA);

// Player A has five  strategies"
PlayerStrategy replayD = factory.createPlayerStrategy();
replayD.setDisplayName("Replay Def.");
replayD.setUID(rID.getRandomIDForSession());
replayD.setHidden(hideReplayD);
replayD.setPlayer(pA);
pA.getPlayerStrategy().add(replayD);

PlayerStrategy eavesD = factory.createPlayerStrategy();
eavesD.setDisplayName("Eavesdropping Def.");
eavesD.setUID(rID.getRandomIDForSession());
eavesD.setHidden(hideEavesD);
eavesD.setPlayer(pA);
pA.getPlayerStrategy().add(eavesD);

PlayerStrategy dosD = factory.createPlayerStrategy();
dosD.setDisplayName("DOS Def.");
dosD.setUID(rID.getRandomIDForSession());
dosD.setHidden(hideDosD);
dosD.setPlayer(pA);
pA.getPlayerStrategy().add(dosD);

PlayerStrategy byzD = factory.createPlayerStrategy();
byzD.setDisplayName("Byzantine Def.");
byzD.setUID(rID.getRandomIDForSession());
byzD.setHidden(hideByzD);
byzD.setPlayer(pA);
pA.getPlayerStrategy().add(byzD);

PlayerStrategy noD = factory.createPlayerStrategy();
noD.setDisplayName("No Defense");
noD.setUID(rID.getRandomIDForSession());
noD.setHidden(hideNoD);
noD.setPlayer(pA);
pA.getPlayerStrategy().add(noD);


// Create the second player:
Player pB = factory.createPlayer();
pB.setUID(rID.getRandomIDForSession());
pB.setDisplayName("Attacker");
pB.setRowPlayer(reverse);
gameData.getPlayer().add(pB);

// Player B has five  strategies"
PlayerStrategy replayA = factory.createPlayerStrategy();
replayA.setDisplayName("Replay Attack");
replayA.setUID(rID.getRandomIDForSession());
replayA.setHidden(hideReplayA);
```

```
replayA.setPlayer(pB);
pB.getPlayerStrategy().add(replayA);

PlayerStrategy eavesA = factory.createPlayerStrategy();
eavesA.setDisplayName("Eavesdropping");
eavesA.setUID(rID.getRandomIDForSession());
eavesA.setHidden(hideEavesA);
eavesA.setPlayer(pA);
pB.getPlayerStrategy().add(eavesA);

PlayerStrategy dosA = factory.createPlayerStrategy();
dosA.setDisplayName("DOS");
dosA.setUID(rID.getRandomIDForSession());
dosA.setHidden(hideDosA);
dosA.setPlayer(pB);
pB.getPlayerStrategy().add(dosA);

PlayerStrategy byzA = factory.createPlayerStrategy();
byzA.setDisplayName("Byzantine Attack");
byzA.setUID(rID.getRandomIDForSession());
byzA.setHidden(hideByzA);
byzA.setPlayer(pB);
pB.getPlayerStrategy().add(byzA);

PlayerStrategy noA = factory.createPlayerStrategy();
noA.setDisplayName("No Attack");
noA.setUID(rID.getRandomIDForSession());
noA.setHidden(hideNoA);
noA.setPlayer(pB);
pB.getPlayerStrategy().add(noA);

// Setup the opponent strategeies:
// Player 1:  utility  = missionAccomplished - damage - defensiveCost

//replayA
OpponentStrategy osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(replayA);

Outcome oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayD);
oc1. setUtility ("{var_replayD_replayA_missionD} - {var_replayD_replayA_damage} - {var_replayDcost}");

Outcome oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesD);
oc2. setUtility ("{var_eavesD_replayA_missionD}-{var_eavesD_replayA_damage} - {var_eavesDcost}");

Outcome oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosD);
oc3. setUtility ("{var_dosD_replayA_missionD}-{var_dosD_replayA_damage} - {var_dosDcost}");
```

```
Outcome oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzD);
oc4.setUtility("{var_byzD_replayA_missionD}-{var_byzD_replayA_damage} - {var_byzDcost}");

Outcome oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noD);
oc5.setUtility("{var_noD_replayA_missionD}-{var_noD_replayA_damage} - {var_noDcost}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pA.getOpponentStrategy().add(osreplayA);

//eavesA
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(eavesA);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayD);
oc1.setUtility("{var_replayD_eavesA_missionD}-{var_replayD_eavesA_damage} - {var_replayDcost}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesD);
oc2.setUtility("{var_eavesD_eavesA_missionD}-{var_eavesD_eavesA_damage} - {var_eavesDcost}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosD);
oc3.setUtility("{var_dosD_eavesA_missionD}-{var_dosD_eavesA_damage} - {var_dosDcost}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzD);
oc4.setUtility("{var_byzD_eavesA_missionD}-{var_byzD_eavesA_damage} - {var_byzDcost}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noD);
oc5.setUtility("{var_noD_eavesA_missionD}-{var_noD_eavesA_damage} - {var_noDcost}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
```

```
pA.getOpponentStrategy().add(osreplayA);

//dosA
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(dosA);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayD);
oc1.setUtility("{var_replayD_dosA_missionD}-{var_replayD_dosA_damage} - {var_replayDcost}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesD);
oc2.setUtility("{var_eavesD_dosA_missionD}-{var_eavesD_dosA_damage} - {var_eavesDcost}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosD);
oc3.setUtility("{var_dosD_dosA_missionD}-{var_dosD_dosA_damage} - {var_dosDcost}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzD);
oc4.setUtility("{var_byzD_dosA_missionD}-{var_byzD_dosA_damage} - {var_byzDcost}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noD);
oc5.setUtility("{var_noD_dosA_missionD}-{var_noD_dosA_damage} - {var_noDcost}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pA.getOpponentStrategy().add(osreplayA);

//byzA
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(byzA);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayD);
oc1.setUtility("{var_replayD_byzA_missionD}-{var_replayD_byzA_damage} - {var_replayDcost}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesD);
```

```
oc2.setUtility("{var_eavesD_byzA_missionD}-{var_eavesD_byzA_damage} - {var_eavesDcost}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosD);
oc3.setUtility("{var_dosD_byzA_missionD}-{var_dosD_byzA_damage} - {var_dosDcost}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzD);
oc4.setUtility("{var_byzD_byzA_missionD}-{var_byzD_byzA_damage} - {var_byzDcost}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noD);
oc5.setUtility("{var_noD_byzA_missionD}-{var_noD_byzA_damage} - {var_noDcost}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pA.getOpponentStrategy().add(osreplayA);


//noA
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(noA);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayD);
oc1.setUtility("{var_replayD_noA_missionD}-{var_replayD_noA_damage} - {var_replayDcost}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesD);
oc2.setUtility("{var_eavesD_noA_missionD}-{var_eavesD_noA_damage} - {var_eavesDcost}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosD);
oc3.setUtility("{var_dosD_noA_missionD}-{var_dosD_noA_damage} - {var_dosDcost}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzD);
oc4.setUtility("{var_byzD_noA_missionD}-{var_byzD_noA_damage} - {var_byzDcost}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noD);
```

```
oc5. setUtility ("{var_noD_noA_missionD}-{var_noD_noA_damage} - {var_noDcost}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pA.getOpponentStrategy().add(osreplayA);


// Player 2: utility = damage - cost - missionFailed
//replayD
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(replayD);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayA);
oc1. setUtility ("{var_replayD_replayA_damage} - {var_replayAcost}-{var_replayD_replayA_missionA}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesA);
oc2. setUtility ("{var_replayD_eavesA_damage} - {var_eavesAcost}-{var_replayD_eavesA_missionA}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosA);
oc3. setUtility ("{var_replayD_dosA_damage} - {var_dosAcost}-{var_replayD_dosA_missionA}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzA);
oc4. setUtility ("{var_replayD_byzA_damage} - {var_byzAcost}-{var_replayD_byzA_missionA}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noA);
oc5. setUtility ("{var_replayD_noA_damage} - {var_noAcost}-{var_replayD_noA_missionA}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pB.getOpponentStrategy().add(osreplayA);

//eavesD
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(eavesD);
```

```
oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayA);
oc1.setUtility("{var_eavesD_replayA_damage} - {var_replayAcost}-{var_eavesD_replayA_missionA}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesA);
oc2.setUtility("{var_eavesD_eavesA_damage} - {var_eavesAcost}-{var_eavesD_eavesA_missionA}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosA);
oc3.setUtility("{var_eavesD_dosA_damage} - {var_dosAcost}-{var_eavesD_dosA_missionA}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzA);
oc4.setUtility("{var_eavesD_byzA_damage} - {var_byzAcost}-{var_eavesD_byzA_missionA}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noA);
oc5.setUtility("{var_eavesD_noA_damage} - {var_noAcost}-{var_eavesD_noA_missionA}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pB.getOpponentStrategy().add(osreplayA);

//dosD
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(dosD);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayA);
oc1.setUtility("{var_dosD_replayA_damage} - {var_replayAcost}-{var_dosD_replayA_missionA}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesA);
oc2.setUtility("{var_dosD_eavesA_damage} - {var_eavesAcost}-{var_dosD_eavesA_missionA}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosA);
oc3.setUtility("{var_dosD_dosA_damage} - {var_dosAcost}-{var_dosD_dosA_missionA}");
```

```
oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzA);
oc4.setUtility("{var_dosD_byzA_damage} - {var_byzAcost}-{var_dosD_byzA_missionA}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noA);
oc5.setUtility("{var_dosD_noA_damage} - {var_noAcost}-{var_dosD_noA_missionA}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pB.getOpponentStrategy().add(osreplayA);

//byzD
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(byzD);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayA);
oc1.setUtility("{var_byzD_replayA_damage} - {var_replayAcost}-{var_byzD_replayA_missionA}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesA);
oc2.setUtility("{var_byzD_eavesA_damage} - {var_eavesAcost}-{var_byzD_eavesA_missionA}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosA);
oc3.setUtility("{var_byzD_dosA_damage} - {var_dosAcost}-{var_byzD_dosA_missionA}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzA);
oc4.setUtility("{var_byzD_byzA_damage} - {var_byzAcost}-{var_byzD_byzA_missionA}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noA);
oc5.setUtility("{var_byzD_noA_damage} - {var_noAcost}-{var_byzD_noA_missionA}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pB.getOpponentStrategy().add(osreplayA);
```

```
//noD
osreplayA = factory.createOpponentStrategy();
osreplayA.setUID(rID.getRandomIDForSession());
osreplayA.setHidden(false);
osreplayA.setPlayerStrategy(noD);

oc1 = factory.createOutcome();
oc1.setHidden(false);
oc1.setPlayerStrategy(replayA);
oc1.setUtility("{var_noD_replayA_damage} - {var_replayAcost}-{var_noD_replayA_missionA}");

oc2 = factory.createOutcome();
oc2.setHidden(false);
oc2.setPlayerStrategy(eavesA);
oc2.setUtility("{var_noD_eavesA_damage} - {var_eavesAcost}-{var_noD_eavesA_missionA}");

oc3 = factory.createOutcome();
oc3.setHidden(false);
oc3.setPlayerStrategy(dosA);
oc3.setUtility("{var_noD_dosA_damage} - {var_dosAcost}-{var_noD_dosA_missionA}");

oc4 = factory.createOutcome();
oc4.setHidden(false);
oc4.setPlayerStrategy(byzA);
oc4.setUtility("{var_noD_byzA_damage} - {var_byzAcost}-{var_noD_byzA_missionA}");

oc5 = factory.createOutcome();
oc5.setHidden(false);
oc5.setPlayerStrategy(noA);
oc5.setUtility("{var_noD_noA_damage} - {var_noAcost}-{var_noD_noA_missionA}");

osreplayA.getOutcome().add(oc1);
osreplayA.getOutcome().add(oc2);
osreplayA.getOutcome().add(oc3);
osreplayA.getOutcome().add(oc4);
osreplayA.getOutcome().add(oc5);
pB.getOpponentStrategy().add(osreplayA);


//COST VARIABLES------------------------------------------------------------

// Create variable 'Replay Cost' and add to game data:
Variable var_replayAcost = factory.createVariable();
var_replayAcost.setUID(rID.getRandomIDForSession());
var_replayAcost.setDisplayName("var_replayAcost");
var_replayAcost.setCurrentValue(BigDecimal.valueOf(4.0));
var_replayAcost.setMinValue(BigDecimal.valueOf(0.0));
var_replayAcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_replayAcost);

Variable var_replayDcost = factory.createVariable();
var_replayDcost.setUID(rID.getRandomIDForSession());
```

```
var_replayDcost.setDisplayName("var_replayDcost");
var_replayDcost.setCurrentValue(BigDecimal.valueOf(4.0));
var_replayDcost.setMinValue(BigDecimal.valueOf(0.0));
var_replayDcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_replayDcost);

// Create variable 'Eavesdropping cost' and add to game data:
Variable var_eavesAcost = factory.createVariable();
var_eavesAcost.setUID(rID.getRandomIDForSession());
var_eavesAcost.setDisplayName("var_eavesAcost");
var_eavesAcost.setCurrentValue(BigDecimal.valueOf(3.0));
var_eavesAcost.setMinValue(BigDecimal.valueOf(0.0));
var_eavesAcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_eavesAcost);

Variable var_eavesDcost = factory.createVariable();
var_eavesDcost.setUID(rID.getRandomIDForSession());
var_eavesDcost.setDisplayName("var_eavesDcost");
var_eavesDcost.setCurrentValue(BigDecimal.valueOf(3.0));
var_eavesDcost.setMinValue(BigDecimal.valueOf(0.0));
var_eavesDcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_eavesDcost);

// Create variable 'DOS cost' and add to game data:
Variable var_dosAcost = factory.createVariable();
var_dosAcost.setUID(rID.getRandomIDForSession());
var_dosAcost.setDisplayName("var_dosAcost");
var_dosAcost.setCurrentValue(BigDecimal.valueOf(2.0));
var_dosAcost.setMinValue(BigDecimal.valueOf(0.0));
var_dosAcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_dosAcost);

Variable var_dosDcost = factory.createVariable();
var_dosDcost.setUID(rID.getRandomIDForSession());
var_dosDcost.setDisplayName("var_dosDcost");
var_dosDcost.setCurrentValue(BigDecimal.valueOf(2.0));
var_dosDcost.setMinValue(BigDecimal.valueOf(0.0));
var_dosDcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_dosDcost);


// Create variable 'Byzantine cost' and add to game data:
Variable var_byzAcost = factory.createVariable();
var_byzAcost.setUID(rID.getRandomIDForSession());
var_byzAcost.setDisplayName("var_byzAcost");
var_byzAcost.setCurrentValue(BigDecimal.valueOf(5.0));
var_byzAcost.setMinValue(BigDecimal.valueOf(0.0));
var_byzAcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_byzAcost);

Variable var_byzDcost = factory.createVariable();
var_byzDcost.setUID(rID.getRandomIDForSession());
var_byzDcost.setDisplayName("var_byzDcost");
var_byzDcost.setCurrentValue(BigDecimal.valueOf(5.0));
```

```
var_byzDcost.setMinValue(BigDecimal.valueOf(0.0));
var_byzDcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_byzDcost);


// Create variable 'No action cost' and add to game data:
Variable var_noAcost = factory.createVariable();
var_noAcost.setUID(rID.getRandomIDForSession());
var_noAcost.setDisplayName("var_noAcost");
var_noAcost.setCurrentValue(BigDecimal.valueOf(0.0));
var_noAcost.setMinValue(BigDecimal.valueOf(0.0));
var_noAcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_noAcost);

Variable var_noDcost = factory.createVariable();
var_noDcost.setUID(rID.getRandomIDForSession());
var_noDcost.setDisplayName("var_noDcost");
var_noDcost.setCurrentValue(BigDecimal.valueOf(0.0));
var_noDcost.setMinValue(BigDecimal.valueOf(0.0));
var_noDcost.setMaxValue(BigDecimal.valueOf(10.0));
gameData.getVariable().add(var_noDcost);



//DAMAGE VARAIABLES------------------------------------------------------------


//Damage against replayD
Variable var_replayD_replayA_damage = factory.createVariable();
var_replayD_replayA_damage.setUID(rID.getRandomIDForSession());
var_replayD_replayA_damage.setDisplayName("var_replayD_replayA_damage");
var_replayD_replayA_damage.setCurrentValue(BigDecimal.valueOf(damage[0][0]));
var_replayD_replayA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_replayA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_replayA_damage);

Variable var_replayD_eavesA_damage = factory.createVariable();
var_replayD_eavesA_damage.setUID(rID.getRandomIDForSession());
var_replayD_eavesA_damage.setDisplayName("var_replayD_eavesA_damage");
var_replayD_eavesA_damage.setCurrentValue(BigDecimal.valueOf(damage[0][1]));
var_replayD_eavesA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_eavesA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_eavesA_damage);

Variable var_replayD_dosA_damage = factory.createVariable();
var_replayD_dosA_damage.setUID(rID.getRandomIDForSession());
var_replayD_dosA_damage.setDisplayName("var_replayD_dosA_damage");
var_replayD_dosA_damage.setCurrentValue(BigDecimal.valueOf(damage[0][2]));
var_replayD_dosA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_dosA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_dosA_damage);

Variable var_replayD_byzA_damage = factory.createVariable();
var_replayD_byzA_damage.setUID(rID.getRandomIDForSession());
```

```
var_replayD_byzA_damage.setDisplayName("var_replayD_byzA_damage");
var_replayD_byzA_damage.setCurrentValue(BigDecimal.valueOf(damage[0][3]));
var_replayD_byzA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_byzA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_byzA_damage);

Variable var_replayD_noA_damage = factory.createVariable();
var_replayD_noA_damage.setUID(rID.getRandomIDForSession());
var_replayD_noA_damage.setDisplayName("var_replayD_noA_damage");
var_replayD_noA_damage.setCurrentValue(BigDecimal.valueOf(damage[0][4]));
var_replayD_noA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_noA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_noA_damage);

//Damage against eavesD
Variable var_eavesD_replayA_damage = factory.createVariable();
var_eavesD_replayA_damage.setUID(rID.getRandomIDForSession());
var_eavesD_replayA_damage.setDisplayName("var_eavesD_replayA_damage");
var_eavesD_replayA_damage.setCurrentValue(BigDecimal.valueOf(damage[1][0]));
var_eavesD_replayA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_replayA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_replayA_damage);

Variable var_eavesD_eavesA_damage = factory.createVariable();
var_eavesD_eavesA_damage.setUID(rID.getRandomIDForSession());
var_eavesD_eavesA_damage.setDisplayName("var_eavesD_eavesA_damage");
var_eavesD_eavesA_damage.setCurrentValue(BigDecimal.valueOf(damage[1][1]));
var_eavesD_eavesA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_eavesA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_eavesA_damage);

Variable var_eavesD_dosA_damage = factory.createVariable();
var_eavesD_dosA_damage.setUID(rID.getRandomIDForSession());
var_eavesD_dosA_damage.setDisplayName("var_eavesD_dosA_damage");
var_eavesD_dosA_damage.setCurrentValue(BigDecimal.valueOf(damage[1][2]));
var_eavesD_dosA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_dosA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_dosA_damage);

Variable var_eavesD_byzA_damage = factory.createVariable();
var_eavesD_byzA_damage.setUID(rID.getRandomIDForSession());
var_eavesD_byzA_damage.setDisplayName("var_eavesD_byzA_damage");
var_eavesD_byzA_damage.setCurrentValue(BigDecimal.valueOf(damage[1][3]));
var_eavesD_byzA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_byzA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_byzA_damage);

Variable var_eavesD_noA_damage = factory.createVariable();
var_eavesD_noA_damage.setUID(rID.getRandomIDForSession());
var_eavesD_noA_damage.setDisplayName("var_eavesD_noA_damage");
var_eavesD_noA_damage.setCurrentValue(BigDecimal.valueOf(damage[1][4]));
var_eavesD_noA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_noA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_noA_damage);
```

```
//Damage against dosD
Variable var_dosD_replayA_damage = factory.createVariable();
var_dosD_replayA_damage.setUID(rID.getRandomIDForSession());
var_dosD_replayA_damage.setDisplayName("var_dosD_replayA_damage");
var_dosD_replayA_damage.setCurrentValue(BigDecimal.valueOf(damage[2][0]));
var_dosD_replayA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_replayA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_replayA_damage);

Variable var_dosD_eavesA_damage = factory.createVariable();
var_dosD_eavesA_damage.setUID(rID.getRandomIDForSession());
var_dosD_eavesA_damage.setDisplayName("var_dosD_eavesA_damage");
var_dosD_eavesA_damage.setCurrentValue(BigDecimal.valueOf(damage[2][1]));
var_dosD_eavesA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_eavesA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_eavesA_damage);

Variable var_dosD_dosA_damage = factory.createVariable();
var_dosD_dosA_damage.setUID(rID.getRandomIDForSession());
var_dosD_dosA_damage.setDisplayName("var_dosD_dosA_damage");
var_dosD_dosA_damage.setCurrentValue(BigDecimal.valueOf(damage[2][2]));
var_dosD_dosA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_dosA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_dosA_damage);

Variable var_dosD_byzA_damage = factory.createVariable();
var_dosD_byzA_damage.setUID(rID.getRandomIDForSession());
var_dosD_byzA_damage.setDisplayName("var_dosD_byzA_damage");
var_dosD_byzA_damage.setCurrentValue(BigDecimal.valueOf(damage[2][3]));
var_dosD_byzA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_byzA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_byzA_damage);

Variable var_dosD_noA_damage = factory.createVariable();
var_dosD_noA_damage.setUID(rID.getRandomIDForSession());
var_dosD_noA_damage.setDisplayName("var_dosD_noA_damage");
var_dosD_noA_damage.setCurrentValue(BigDecimal.valueOf(damage[2][4]));
var_dosD_noA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_noA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_noA_damage);


//Damage against byzD
Variable var_byzD_replayA_damage = factory.createVariable();
var_byzD_replayA_damage.setUID(rID.getRandomIDForSession());
var_byzD_replayA_damage.setDisplayName("var_byzD_replayA_damage");
var_byzD_replayA_damage.setCurrentValue(BigDecimal.valueOf(damage[3][0]));
var_byzD_replayA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_replayA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_replayA_damage);

Variable var_byzD_eavesA_damage = factory.createVariable();
```

```
var_byzD_eavesA_damage.setUID(rID.getRandomIDForSession());
var_byzD_eavesA_damage.setDisplayName("var_byzD_eavesA_damage");
var_byzD_eavesA_damage.setCurrentValue(BigDecimal.valueOf(damage[3][1]));
var_byzD_eavesA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_eavesA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_eavesA_damage);

Variable var_byzD_dosA_damage = factory.createVariable();
var_byzD_dosA_damage.setUID(rID.getRandomIDForSession());
var_byzD_dosA_damage.setDisplayName("var_byzD_dosA_damage");
var_byzD_dosA_damage.setCurrentValue(BigDecimal.valueOf(damage[3][2]));
var_byzD_dosA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_dosA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_dosA_damage);

Variable var_byzD_byzA_damage = factory.createVariable();
var_byzD_byzA_damage.setUID(rID.getRandomIDForSession());
var_byzD_byzA_damage.setDisplayName("var_byzD_byzA_damage");
var_byzD_byzA_damage.setCurrentValue(BigDecimal.valueOf(damage[3][3]));
var_byzD_byzA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_byzA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_byzA_damage);

Variable var_byzD_noA_damage = factory.createVariable();
var_byzD_noA_damage.setUID(rID.getRandomIDForSession());
var_byzD_noA_damage.setDisplayName("var_byzD_noA_damage");
var_byzD_noA_damage.setCurrentValue(BigDecimal.valueOf(damage[3][4]));
var_byzD_noA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_noA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_noA_damage);


//Damage against noD
Variable var_noD_replayA_damage = factory.createVariable();
var_noD_replayA_damage.setUID(rID.getRandomIDForSession());
var_noD_replayA_damage.setDisplayName("var_noD_replayA_damage");
var_noD_replayA_damage.setCurrentValue(BigDecimal.valueOf(damage[4][0]));
var_noD_replayA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_noD_replayA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_replayA_damage);

Variable var_noD_eavesA_damage = factory.createVariable();
var_noD_eavesA_damage.setUID(rID.getRandomIDForSession());
var_noD_eavesA_damage.setDisplayName("var_noD_eavesA_damage");
var_noD_eavesA_damage.setCurrentValue(BigDecimal.valueOf(damage[4][1]));
var_noD_eavesA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_noD_eavesA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_eavesA_damage);

Variable var_noD_dosA_damage = factory.createVariable();
var_noD_dosA_damage.setUID(rID.getRandomIDForSession());
var_noD_dosA_damage.setDisplayName("var_noD_dosA_damage");
var_noD_dosA_damage.setCurrentValue(BigDecimal.valueOf(damage[4][2]));
var_noD_dosA_damage.setMinValue(BigDecimal.valueOf(0.0));
```

```
var_noD_dosA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_dosA_damage);

Variable var_noD_byzA_damage = factory.createVariable();
var_noD_byzA_damage.setUID(rID.getRandomIDForSession());
var_noD_byzA_damage.setDisplayName("var_noD_byzA_damage");
var_noD_byzA_damage.setCurrentValue(BigDecimal.valueOf(damage[4][3]));
var_noD_byzA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_noD_byzA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_byzA_damage);

Variable var_noD_noA_damage = factory.createVariable();
var_noD_noA_damage.setUID(rID.getRandomIDForSession());
var_noD_noA_damage.setDisplayName("var_noD_noA_damage");
var_noD_noA_damage.setCurrentValue(BigDecimal.valueOf(damage[4][4]));
var_noD_noA_damage.setMinValue(BigDecimal.valueOf(0.0));
var_noD_noA_damage.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_noA_damage);

//MISSION VARIABLES-----------------------------------------------------------

//Mission Accomplished

// Mission Accomplish with replayD
Variable var_replayD_replayA_missionD = factory.createVariable();
var_replayD_replayA_missionD.setUID(rID.getRandomIDForSession());
var_replayD_replayA_missionD.setDisplayName("var_replayD_replayA_missionD");
var_replayD_replayA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[0][0]));
var_replayD_replayA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_replayA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_replayA_missionD);

Variable var_replayD_eavesA_missionD = factory.createVariable();
var_replayD_eavesA_missionD.setUID(rID.getRandomIDForSession());
var_replayD_eavesA_missionD.setDisplayName("var_replayD_eavesA_missionD");
var_replayD_eavesA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[0][1]));
var_replayD_eavesA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_eavesA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_eavesA_missionD);

Variable var_replayD_dosA_missionD = factory.createVariable();
var_replayD_dosA_missionD.setUID(rID.getRandomIDForSession());
var_replayD_dosA_missionD.setDisplayName("var_replayD_dosA_missionD");
var_replayD_dosA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[0][2]));
var_replayD_dosA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_dosA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_dosA_missionD);

Variable var_replayD_byzA_missionD = factory.createVariable();
var_replayD_byzA_missionD.setUID(rID.getRandomIDForSession());
var_replayD_byzA_missionD.setDisplayName("var_replayD_byzA_missionD");
var_replayD_byzA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[0][3]));
var_replayD_byzA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_byzA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
```

```
gameData.getVariable().add(var_replayD_byzA_missionD);

Variable var_replayD_noA_missionD = factory.createVariable();
var_replayD_noA_missionD.setUID(rID.getRandomIDForSession());
var_replayD_noA_missionD.setDisplayName("var_replayD_noA_missionD");
var_replayD_noA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[0][4]));
var_replayD_noA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_noA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_noA_missionD);

//Mission Accomplish with eavesD
Variable var_eavesD_replayA_missionD = factory.createVariable();
var_eavesD_replayA_missionD.setUID(rID.getRandomIDForSession());
var_eavesD_replayA_missionD.setDisplayName("var_eavesD_replayA_missionD");
var_eavesD_replayA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[1][0]));
var_eavesD_replayA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_replayA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_replayA_missionD);

Variable var_eavesD_eavesA_missionD = factory.createVariable();
var_eavesD_eavesA_missionD.setUID(rID.getRandomIDForSession());
var_eavesD_eavesA_missionD.setDisplayName("var_eavesD_eavesA_missionD");
var_eavesD_eavesA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[1][1]));
var_eavesD_eavesA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_eavesA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_eavesA_missionD);

Variable var_eavesD_dosA_missionD = factory.createVariable();
var_eavesD_dosA_missionD.setUID(rID.getRandomIDForSession());
var_eavesD_dosA_missionD.setDisplayName("var_eavesD_dosA_missionD");
var_eavesD_dosA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[1][2]));
var_eavesD_dosA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_dosA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_dosA_missionD);

Variable var_eavesD_byzA_missionD = factory.createVariable();
var_eavesD_byzA_missionD.setUID(rID.getRandomIDForSession());
var_eavesD_byzA_missionD.setDisplayName("var_eavesD_byzA_missionD");
var_eavesD_byzA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[1][3]));
var_eavesD_byzA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_byzA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_byzA_missionD);

Variable var_eavesD_noA_missionD = factory.createVariable();
var_eavesD_noA_missionD.setUID(rID.getRandomIDForSession());
var_eavesD_noA_missionD.setDisplayName("var_eavesD_noA_missionD");
var_eavesD_noA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[1][4]));
var_eavesD_noA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_noA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_noA_missionD);


//Mission Accomplish with dosD
Variable var_dosD_replayA_missionD = factory.createVariable();
```

```
var_dosD_replayA_missionD.setUID(rID.getRandomIDForSession());
var_dosD_replayA_missionD.setDisplayName("var_dosD_replayA_missionD");
var_dosD_replayA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[2][0]));
var_dosD_replayA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_replayA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_replayA_missionD);

Variable var_dosD_eavesA_missionD = factory.createVariable();
var_dosD_eavesA_missionD.setUID(rID.getRandomIDForSession());
var_dosD_eavesA_missionD.setDisplayName("var_dosD_eavesA_missionD");
var_dosD_eavesA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[2][1]));
var_dosD_eavesA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_eavesA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_eavesA_missionD);

Variable var_dosD_dosA_missionD = factory.createVariable();
var_dosD_dosA_missionD.setUID(rID.getRandomIDForSession());
var_dosD_dosA_missionD.setDisplayName("var_dosD_dosA_missionD");
var_dosD_dosA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[2][2]));
var_dosD_dosA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_dosA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_dosA_missionD);

Variable var_dosD_byzA_missionD = factory.createVariable();
var_dosD_byzA_missionD.setUID(rID.getRandomIDForSession());
var_dosD_byzA_missionD.setDisplayName("var_dosD_byzA_missionD");
var_dosD_byzA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[2][3]));
var_dosD_byzA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_byzA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_byzA_missionD);

Variable var_dosD_noA_missionD = factory.createVariable();
var_dosD_noA_missionD.setUID(rID.getRandomIDForSession());
var_dosD_noA_missionD.setDisplayName("var_dosD_noA_missionD");
var_dosD_noA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[2][4]));
var_dosD_noA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_noA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_noA_missionD);


//Mission Accomplish with byzD
Variable var_byzD_replayA_missionD = factory.createVariable();
var_byzD_replayA_missionD.setUID(rID.getRandomIDForSession());
var_byzD_replayA_missionD.setDisplayName("var_byzD_replayA_missionD");
var_byzD_replayA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[3][0]));
var_byzD_replayA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_replayA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_replayA_missionD);

Variable var_byzD_eavesA_missionD = factory.createVariable();
var_byzD_eavesA_missionD.setUID(rID.getRandomIDForSession());
var_byzD_eavesA_missionD.setDisplayName("var_byzD_eavesA_missionD");
var_byzD_eavesA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[3][1]));
var_byzD_eavesA_missionD.setMinValue(BigDecimal.valueOf(0.0));
```

```
var_byzD_eavesA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_eavesA_missionD);

Variable var_byzD_dosA_missionD = factory.createVariable();
var_byzD_dosA_missionD.setUID(rID.getRandomIDForSession());
var_byzD_dosA_missionD.setDisplayName("var_byzD_dosA_missionD");
var_byzD_dosA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[3][2]));
var_byzD_dosA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_dosA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_dosA_missionD);

Variable var_byzD_byzA_missionD = factory.createVariable();
var_byzD_byzA_missionD.setUID(rID.getRandomIDForSession());
var_byzD_byzA_missionD.setDisplayName("var_byzD_byzA_missionD");
var_byzD_byzA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[3][3]));
var_byzD_byzA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_byzA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_byzA_missionD);

Variable var_byzD_noA_missionD = factory.createVariable();
var_byzD_noA_missionD.setUID(rID.getRandomIDForSession());
var_byzD_noA_missionD.setDisplayName("var_byzD_noA_missionD");
var_byzD_noA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[3][4]));
var_byzD_noA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_noA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_noA_missionD);


//Mission Accomplish with noD
Variable var_noD_replayA_missionD = factory.createVariable();
var_noD_replayA_missionD.setUID(rID.getRandomIDForSession());
var_noD_replayA_missionD.setDisplayName("var_noD_replayA_missionD");
var_noD_replayA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[4][0]));
var_noD_replayA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_noD_replayA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_replayA_missionD);

Variable var_noD_eavesA_missionD = factory.createVariable();
var_noD_eavesA_missionD.setUID(rID.getRandomIDForSession());
var_noD_eavesA_missionD.setDisplayName("var_noD_eavesA_missionD");
var_noD_eavesA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[4][1]));
var_noD_eavesA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_noD_eavesA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_eavesA_missionD);

Variable var_noD_dosA_missionD = factory.createVariable();
var_noD_dosA_missionD.setUID(rID.getRandomIDForSession());
var_noD_dosA_missionD.setDisplayName("var_noD_dosA_missionD");
var_noD_dosA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[4][2]));
var_noD_dosA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_noD_dosA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_dosA_missionD);

Variable var_noD_byzA_missionD = factory.createVariable();
```

```
var_noD_byzA_missionD.setUID(rID.getRandomIDForSession());
var_noD_byzA_missionD.setDisplayName("var_noD_byzA_missionD");
var_noD_byzA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[4][3]));
var_noD_byzA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_noD_byzA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_byzA_missionD);

Variable var_noD_noA_missionD = factory.createVariable();
var_noD_noA_missionD.setUID(rID.getRandomIDForSession());
var_noD_noA_missionD.setDisplayName("var_noD_noA_missionD");
var_noD_noA_missionD.setCurrentValue(BigDecimal.valueOf(missionAccomplished[4][4]));
var_noD_noA_missionD.setMinValue(BigDecimal.valueOf(0.0));
var_noD_noA_missionD.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_noA_missionD);


//Mission Failed

// Mission Failed with replayD
Variable var_replayD_replayA_missionA = factory.createVariable();
var_replayD_replayA_missionA.setUID(rID.getRandomIDForSession());
var_replayD_replayA_missionA.setDisplayName("var_replayD_replayA_missionA");
var_replayD_replayA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[0][0]));
var_replayD_replayA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_replayA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_replayA_missionA);

Variable var_replayD_eavesA_missionA = factory.createVariable();
var_replayD_eavesA_missionA.setUID(rID.getRandomIDForSession());
var_replayD_eavesA_missionA.setDisplayName("var_replayD_eavesA_missionA");
var_replayD_eavesA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[0][1]));
var_replayD_eavesA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_eavesA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_eavesA_missionA);

Variable var_replayD_dosA_missionA = factory.createVariable();
var_replayD_dosA_missionA.setUID(rID.getRandomIDForSession());
var_replayD_dosA_missionA.setDisplayName("var_replayD_dosA_missionA");
var_replayD_dosA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[0][2]));
var_replayD_dosA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_dosA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_dosA_missionA);

Variable var_replayD_byzA_missionA = factory.createVariable();
var_replayD_byzA_missionA.setUID(rID.getRandomIDForSession());
var_replayD_byzA_missionA.setDisplayName("var_replayD_byzA_missionA");
var_replayD_byzA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[0][3]));
var_replayD_byzA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_byzA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_byzA_missionA);

Variable var_replayD_noA_missionA = factory.createVariable();
var_replayD_noA_missionA.setUID(rID.getRandomIDForSession());
var_replayD_noA_missionA.setDisplayName("var_replayD_noA_missionA");
```

```
var_replayD_noA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[0][4]));
var_replayD_noA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_replayD_noA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_replayD_noA_missionA);

//Mission Failed with eavesD
Variable var_eavesD_replayA_missionA = factory.createVariable();
var_eavesD_replayA_missionA.setUID(rID.getRandomIDForSession());
var_eavesD_replayA_missionA.setDisplayName("var_eavesD_replayA_missionA");
var_eavesD_replayA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[1][0]));
var_eavesD_replayA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_replayA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_replayA_missionA);

Variable var_eavesD_eavesA_missionA = factory.createVariable();
var_eavesD_eavesA_missionA.setUID(rID.getRandomIDForSession());
var_eavesD_eavesA_missionA.setDisplayName("var_eavesD_eavesA_missionA");
var_eavesD_eavesA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[1][1]));
var_eavesD_eavesA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_eavesA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_eavesA_missionA);

Variable var_eavesD_dosA_missionA = factory.createVariable();
var_eavesD_dosA_missionA.setUID(rID.getRandomIDForSession());
var_eavesD_dosA_missionA.setDisplayName("var_eavesD_dosA_missionA");
var_eavesD_dosA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[1][2]));
var_eavesD_dosA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_dosA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_dosA_missionA);

Variable var_eavesD_byzA_missionA = factory.createVariable();
var_eavesD_byzA_missionA.setUID(rID.getRandomIDForSession());
var_eavesD_byzA_missionA.setDisplayName("var_eavesD_byzA_missionA");
var_eavesD_byzA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[1][3]));
var_eavesD_byzA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_byzA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_byzA_missionA);

Variable var_eavesD_noA_missionA = factory.createVariable();
var_eavesD_noA_missionA.setUID(rID.getRandomIDForSession());
var_eavesD_noA_missionA.setDisplayName("var_eavesD_noA_missionA");
var_eavesD_noA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[1][4]));
var_eavesD_noA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_eavesD_noA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_eavesD_noA_missionA);


//Mission Failed with dosD
Variable var_dosD_replayA_missionA = factory.createVariable();
var_dosD_replayA_missionA.setUID(rID.getRandomIDForSession());
var_dosD_replayA_missionA.setDisplayName("var_dosD_replayA_missionA");
var_dosD_replayA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[2][0]));
var_dosD_replayA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_replayA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
```

```
gameData.getVariable().add(var_dosD_replayA_missionA);

Variable var_dosD_eavesA_missionA = factory.createVariable();
var_dosD_eavesA_missionA.setUID(rID.getRandomIDForSession());
var_dosD_eavesA_missionA.setDisplayName("var_dosD_eavesA_missionA");
var_dosD_eavesA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[2][1]));
var_dosD_eavesA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_eavesA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_eavesA_missionA);

Variable var_dosD_dosA_missionA = factory.createVariable();
var_dosD_dosA_missionA.setUID(rID.getRandomIDForSession());
var_dosD_dosA_missionA.setDisplayName("var_dosD_dosA_missionA");
var_dosD_dosA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[2][2]));
var_dosD_dosA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_dosA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_dosA_missionA);

Variable var_dosD_byzA_missionA = factory.createVariable();
var_dosD_byzA_missionA.setUID(rID.getRandomIDForSession());
var_dosD_byzA_missionA.setDisplayName("var_dosD_byzA_missionA");
var_dosD_byzA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[2][3]));
var_dosD_byzA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_byzA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_byzA_missionA);

Variable var_dosD_noA_missionA = factory.createVariable();
var_dosD_noA_missionA.setUID(rID.getRandomIDForSession());
var_dosD_noA_missionA.setDisplayName("var_dosD_noA_missionA");
var_dosD_noA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[2][4]));
var_dosD_noA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_dosD_noA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_dosD_noA_missionA);


//Mission Failed with byzD
Variable var_byzD_replayA_missionA = factory.createVariable();
var_byzD_replayA_missionA.setUID(rID.getRandomIDForSession());
var_byzD_replayA_missionA.setDisplayName("var_byzD_replayA_missionA");
var_byzD_replayA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[3][0]));
var_byzD_replayA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_replayA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_replayA_missionA);

Variable var_byzD_eavesA_missionA = factory.createVariable();
var_byzD_eavesA_missionA.setUID(rID.getRandomIDForSession());
var_byzD_eavesA_missionA.setDisplayName("var_byzD_eavesA_missionA");
var_byzD_eavesA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[3][1]));
var_byzD_eavesA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_eavesA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_eavesA_missionA);

Variable var_byzD_dosA_missionA = factory.createVariable();
var_byzD_dosA_missionA.setUID(rID.getRandomIDForSession());
```

```
var_byzD_dosA_missionA.setDisplayName("var_byzD_dosA_missionA");
var_byzD_dosA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[3][2]));
var_byzD_dosA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_dosA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_dosA_missionA);

Variable var_byzD_byzA_missionA = factory.createVariable();
var_byzD_byzA_missionA.setUID(rID.getRandomIDForSession());
var_byzD_byzA_missionA.setDisplayName("var_byzD_byzA_missionA");
var_byzD_byzA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[3][3]));
var_byzD_byzA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_byzA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_byzA_missionA);

Variable var_byzD_noA_missionA = factory.createVariable();
var_byzD_noA_missionA.setUID(rID.getRandomIDForSession());
var_byzD_noA_missionA.setDisplayName("var_byzD_noA_missionA");
var_byzD_noA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[3][4]));
var_byzD_noA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_byzD_noA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_byzD_noA_missionA);


//Mission Failed with noD
Variable var_noD_replayA_missionA = factory.createVariable();
var_noD_replayA_missionA.setUID(rID.getRandomIDForSession());
var_noD_replayA_missionA.setDisplayName("var_noD_replayA_missionA");
var_noD_replayA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[4][0]));
var_noD_replayA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_noD_replayA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_replayA_missionA);

Variable var_noD_eavesA_missionA = factory.createVariable();
var_noD_eavesA_missionA.setUID(rID.getRandomIDForSession());
var_noD_eavesA_missionA.setDisplayName("var_noD_eavesA_missionA");
var_noD_eavesA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[4][1]));
var_noD_eavesA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_noD_eavesA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_eavesA_missionA);

Variable var_noD_dosA_missionA = factory.createVariable();
var_noD_dosA_missionA.setUID(rID.getRandomIDForSession());
var_noD_dosA_missionA.setDisplayName("var_noD_dosA_missionA");
var_noD_dosA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[4][2]));
var_noD_dosA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_noD_dosA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_dosA_missionA);

Variable var_noD_byzA_missionA = factory.createVariable();
var_noD_byzA_missionA.setUID(rID.getRandomIDForSession());
var_noD_byzA_missionA.setDisplayName("var_noD_byzA_missionA");
var_noD_byzA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[4][3]));
var_noD_byzA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_noD_byzA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
```

```
gameData.getVariable().add(var_noD_byzA_missionA);

Variable var_noD_noA_missionA = factory.createVariable();
var_noD_noA_missionA.setUID(rID.getRandomIDForSession());
var_noD_noA_missionA.setDisplayName("var_noD_noA_missionA");
var_noD_noA_missionA.setCurrentValue(BigDecimal.valueOf(missionFailed[4][4]));
var_noD_noA_missionA.setMinValue(BigDecimal.valueOf(0.0));
var_noD_noA_missionA.setMaxValue(BigDecimal.valueOf(12.0));
gameData.getVariable().add(var_noD_noA_missionA);


return gameData;
}

}
```

Appendix C. MIP Implementation

```
/*This code finds the nash equilibrium utilizing MIP. The cplex library is utilized
* to find the optimal solutions. The code actually finds the best solution.
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package com.nicholaskovach.jhalf.components.equilibriums.nash;
import com.nicholaskovach.jhalf.components.interfaces.Hypergame;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.Player;
import com.nicholaskovach.jhalf.components.interfaces.jaxb.PlayerStrategy;


import ilog.concert.*;
import ilog.cplex.*;

/**
*
* @author youkn
*/
public class MIP {

  public static double[][] findEquilibriumsMIP(Hypergame hypergame){

    Player rowPlayer = hypergame.getRowPlayer();
    Player colPlayer = hypergame.getColumnPlayer();

    int m = rowPlayer.getPlayerStrategy().size();
    int n = colPlayer.getPlayerStrategy().size();

    double[][] utilitiesA = new double[m][n];
    double[][] utilitiesB = new double[n][m];

    double max = -Double.MAX_VALUE;
    double min = Double.MAX_VALUE;

    int i = 0;
    for (PlayerStrategy rowStrategy : rowPlayer.getPlayerStrategy()) {
      int j = 0;
      for (PlayerStrategy colStrategy : colPlayer.getPlayerStrategy()) {
        utilitiesA[i][j] = hypergame.getPlayerEvaluatedUtility(rowPlayer, rowStrategy, colStrategy);

        if(utilitiesA[i][j]>max){
          max = utilitiesA[i][j];
        }

        if(utilitiesA[i][j]<min){
          min = utilitiesA[i][j];
        }

        j++;
      }
      i++;
```

```
}

double Ua = max - min;

max = -Double.MAX_VALUE;
min = Double.MAX_VALUE;

i = 0;
for (PlayerStrategy colStrategy : colPlayer.getPlayerStrategy()) {
  int j = 0;
  for (PlayerStrategy rowStrategy : rowPlayer.getPlayerStrategy()) {
    utilitiesB[i][j] = hypergame.getPlayerEvaluatedUtility(colPlayer, colStrategy, rowStrategy);

    if(utilitiesB[i][j]>max){
      max = utilitiesB[i][j];
    }

    if(utilitiesB[i][j]<min){
      min = utilitiesB[i][j];
    }

    j++;
  }
  i++;
}

double Ub = max - min;


try{
  IloCplex cplex = new IloCplex();

  //variables
  IloNumVar[] pA = cplex.numVarArray(utilitiesA.length, 0, 1);
  IloNumVar ua = cplex.numVar(-Double.MAX_VALUE,Double.MAX_VALUE, "ua");
  IloNumVar[] usa = cplex.numVarArray(utilitiesA.length, -Double.MAX_VALUE, Double.MAX_VALUE);
  IloNumVar[] rsa = cplex.numVarArray(utilitiesA.length, -Double.MAX_VALUE, Double.MAX_VALUE);
  IloIntVar[] bsa = cplex.boolVarArray(utilitiesA.length);

  IloNumVar[] pB = cplex.numVarArray(utilitiesB.length, 0, 1);
  IloNumVar ub = cplex.numVar(-Double.MAX_VALUE,Double.MAX_VALUE, "ua");
  IloNumVar[] usb = cplex.numVarArray(utilitiesB.length, -Double.MAX_VALUE, Double.MAX_VALUE);
  IloNumVar[] rsb = cplex.numVarArray(utilitiesB.length, -Double.MAX_VALUE, Double.MAX_VALUE);
  IloIntVar[] bsb = cplex.boolVarArray(utilitiesB.length);

  //add objective
  //cplex.addMaximize(cplex.sum(bsb));//(minimize support)
  //cplex.addMinimize(cplex.sum(bsb));//(maximize support)


  //define constraints
  cplex.addEq(cplex.sum(pA),1);
  cplex.addEq(cplex.sum(pB),1);
```

```
for(int g = 0;g<utilitiesA.length;g++){
  cplex.addEq(cplex.scalProd(utilitiesA[g], pB), usa[g]);

  cplex.addGe(ua,usa[g]);

  cplex.addEq(rsa[g],cplex.diff(ua,usa[g]));

  cplex.addLe(pA[g],cplex.diff(1,bsa[g]));

  cplex.addLe(rsa[g], cplex.prod(Ua, bsa[g]));
}

for(int g = 0;g<utilitiesB.length;g++){
  cplex.addEq(cplex.scalProd(utilitiesB[g], pA), usb[g]);

  cplex.addGe(ub,usb[g]);

  cplex.addEq(rsb[g],cplex.diff(ub,usb[g]));

  cplex.addLe(pB[g],cplex.diff(1,bsb[g]));

  cplex.addLe(rsb[g], cplex.prod(Ub, bsb[g]));
}
//
if(cplex.populate()){


  /*To find different equilibriums change the h value in both place of the loop to a fixed number
  For example the following code

  for(int k = 0;k<m;k++){
    equilibriums[h][k] = cplex.getValue(pA[k], 1);
  }

  for(int k = m;k<(m+n);k++){
    equilibriums[h][k] = cplex.getValue(pB[k-m], 1);

  }

  will find the second equilibrium in the pool, because "0" is the first. The second equilibrium will appear in all the
  the arrays as it is a fixed value and that is the equilibrium that will be in HAT.
  To instead find all the equilibriums set it back to h, but this will only show the first equilibrium
  in HAT.
  */
  int totaleq = cplex.getSolnPoolNsolns();
  double [][] equilibriums = new double[totaleq][m+n];

  for(int h=0;h<equilibriums.length;h++){

    for(int k = 0;k<m;k++){
      equilibriums[h][k] = cplex.getValue(pA[k], h);//7
    }

    for(int k = m;k<(m+n);k++){
```

```
            equilibriums[h][k] = cplex.getValue(pB[k-m], h);//7

          }
        }
        return equilibriums;
      }

      else{
        System.out.println("Model not solved");
      }
    }catch(IloException exc){

  }
  double [][] noSol = new double[1][1];
  return noSol;
}

}
```

Bibliography

1. NCCIC, "Grizzly steppe russian malicious cyber activity," https://www.us-cert.gov/sites/default/files/publications/JAR_16-20296A_GRIZZLY\%20STEPPE-2016-1229.pdf, Dec 2016.

2. Barbara Starr, Jim Sciutto, and Ryan Browne, "Adviser contradicts trump: Russians hacked the us," http://www.cnn.com/2017/01/02/politics/digital-fingerprints-russia-hacking, Jan 2017.

3. David Bellhouse, "The problem of waldegrave.," *Journal lectronique d'Histoire des Probabilits et de la Statistique [electronic only]*, vol. 3, no. 2, pp. Article 1, 12 p., electronic only–Article 1, 12 p., electronic only, 2007.

4. John V Neumann, *On the Theory of Games of Strategy*, vol. 4, pp. 13–42, Princeton University Press, 1959.

5. Chelsea Whyte, "Game theory says publicly shaming cyberattackers could backfire," https://www.newscientist.com/article/2122909-game-theory-says-publicly-shaming-cyberattackers-could-backfire/, Feb 2017.

6. P. G. Bennett and M. R. Dando, "Complex strategic analysis: A hypergame study of the fall of france," *The Journal of the Operational Research Society*, vol. 30, no. 1, pp. 23–32, 1979.

7. P. G. Bennett, M. R. Dando, and R. G. Sharp, "Using hypergames to model difficult social issues: An approach to the case of soccer hooliganism," *The Journal of the Operational Research Society*, vol. 31, no. 7, pp. 621–635, 1980.

8. Maxime Leclerc and Brahim Chaib-draa, "Hypergame analysis in e-commerce: A preliminary report," *CIRANO Working Papers*, 2002.

9. Alan S. Gibson, "Applied hypergame theory for network defense," M.S. thesis, Air Force Institute of Technology, 2013.

10. James Thomas House and George Cybenko, "Hypergame theory applied to cyber attack and defense," *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IX*, 2010.

11. Nicholas S. Kovach, Alan S. Gibson, and Gary B. Lamont, "Hypergame theory: A model for conflict, misperception, and deception," *Game Theory*, vol. 2015, pp. 1–20, 2015.

12. Richard D Mckelvey, Andrew Mclennan, and Theodore L Turocy, "Gambit," http://www.gambit-project.org, 2014.

13. Eugene Nudelman, Kevin Leyton-Brown, Jennifer Wortman, and Yoav Shoham, "Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms," *Autonomous Agents and Multiagent Systems International Joint Conference*, vol. 2, pp. 880–887, 2004.

14. Lachlan Brumley, "Hypant: A hypergame analysis tool," M.S. thesis, Monash University, 2003.

15. Muhong Wang, Keith W. Hipel, and Niall M. Fraser, "Modeling misperceptions in games," *Behavioral Science*, vol. 33, no. 3, pp. 207–223, 1988.

16. Steve Tadelis, *Game theory: an introduction*, Princeton University Press, 2013.

17. Émile Borel, "The theory of play and integral equations with skew symmetric kernels," *Econometrica*, vol. 21, no. 1, pp. 97–100, Jan 1953.

18. Carlton E Lemke and J.T. Howson, "Equilibrium points of bimatrix games," *SIAM Journal on Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.

19. Dave Pritchard, "Game theory and algorithms lecture 6: The lemke-howson ...," `http://ints.io/daveagp/gta/lecture6.pdf`, Mar 2011.

20. G. Van Der Laan, A. J. J. Talman, and L. Van Der Heyden, "Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 377–397, 1987.

21. Charles Audet, Pierre Hansen, Brigitte Jaumard, and Gilles Savard, "Enumeration of all extreme equilibria of bimatrix games," *SIAM Journal on Scientific Computing*, vol. 23, no. 1, pp. 323–338, 2001.

22. Gabriel D Rosenberg, "Enumeration of all extreme equilibria of bimatrix games with integer pivoting and improved degeneracy check," *CDAM*, Dec 2005.

23. Jonathan Widger and Daniel Grosu, "Computing equilibria in bimatrix games by parallel support enumeration," *2008 International Symposium on Parallel and Distributed Computing*, 2008.

24. Srihari Govindan and Robert Wilson, "A global newton method to compute nash equilibria," *Journal of Economic Theory*, vol. 110, no. 1, pp. 65–86, 2003.

25. Srihari Govindan and Robert Wilson, "Computing nash equilibria by iterated polymatrix approximation," *Journal of Economic Dynamics and Control*, vol. 28, no. 7, pp. 1229–1241, 2004.

26. Richard D Mckelvey, "A liapunov function for nash equilibria," *Cal Tech Social Sciences Working Paper 953*, 1991.

27. Daphne Koller, Nimrod Megiddo, and Bernhard Von Stengel, "Efficient computation of equilibria for extensive two-person games," *Games and Economic Behavior*, vol. 14, no. 2, pp. 247–259, 1996.

28. C. E. Lemke, "Bimatrix equilibrium points and mathematical programming," *Management Science*, vol. 11, no. 7, pp. 681–689, 1965.

29. Theodore L. Turocy, "A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence," *Games and Economic Behavior*, vol. 51, no. 2, pp. 243–263, 2005.

30. Ryan Porter, Eugene Nudelman, and Yoav Shoham, "Simple search methods for finding a nash equilibrium," *Games and Economic Behavior*, vol. 63, no. 2, pp. 642–662, 2008.

31. Thomas Sandholm, Andrew Gilpin, and Vincent Conitzer, *Proceedings of the 20th national conference on Artificial intelligence*, vol. 2, pp. 495–501, AAI Press, 2005.

32. Wikipedia, "Game theory — wikipedia, the free encyclopedia," `https://en.wikipedia.org/wiki/Game_theory#Perfect_information_and_imperfect_information`, 2017, [Online; accessed 17-February-2017].

33. Wikipedia, "Chicken (game) — wikipedia, the free encyclopedia," `https://en.wikipedia.org/w/index.php?title=Chicken_(game)&oldid=762598750`, 2017, [Online; accessed 28-February-2017].

34. Peter G. Bennett, "Hypergames: Developing a model of conflict," *Futures*, vol. 12, no. 6, pp. 489–507, 1980.

35. Russell Richardson Vane, *Using hypergames to select plans in competitive environments*, Ph.D. thesis, George Mason University, 2000.

36. Alvaro A. Cardenas, Saurabh Amin, and Shankar Sastry, "Secure control: Towards survivable cyber-physical systems," *2008 The 28th International Conference on Distributed Computing Systems Workshops*, 2008.

37. Hafedh Chourabi, Taewoo Nam, Shawn Walker, J. Ramon Gil-Garcia, Sehl Mellouli, Karine Nahon, Theresa A. Pardo, and Hans Jochen Scholl, "Understanding smart cities: An integrative framework," *2012 45th Hawaii International Conference on System Sciences*, 2012.

38. Anshuman Singh, Arun Lakhotia, and Andrew Walenstein, "Malware antimalware games," 2010.

39. Ming Zhang, Zizhan Zheng, and Ness B. Shroff, "Stealthy attacks and observable defenses: A game theoretic model under strict resource constraints," *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2014.

40. Anibal Sanjab and Walid Saad, "On bounded rationality in cyber-physical systems security: Game-theoretic analysis with application to smart grid protection," *2016 Joint Workshop on Cyber- Physical Security and Resilience in Smart Grids (CPSR-SG)*, 2016.

41. Lin Chen and J. Leneutre, "A game theoretical framework on intrusion detection in heterogeneous networks," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 2, pp. 165–178, 2009.

42. Linda Bushnell, "Passivity-based approach to control and game theoretic cyber defense," Dec 2015.

43. Yu Liu, Cristina Comaniciu, and Hong Man, "A bayesian game approach for intrusion detection in wireless ad hoc networks," *Proceeding from the 2006 workshop on Game theory for communications and networks - GameNets '06*, 2006.

44. Hayreddin Ãeker, Jun Zhuang, Shambhu Upadhyaya, Quang Duy La, and Boon-Hee Soong, "Deception-based game theoretical approach to mitigate dos attacks," *Lecture Notes in Computer Science Decision and Game Theory for Security*, pp. 18–38, 2016.

45. Quanyan Zhu and Tamer Basar, "Dynamic policy-based ids configuration," *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009.

46. Christopher N. Gutierrez, Mohammed H. Almeshekah, Jeff Avery, Saurabh Bagchi, and Eugene H. Spafford, "Modeling deception in information security as a hypergame: A primer," in *Proceedings of the 16th Annual Information Security Symposium*, West Lafayette, IN, 2015, CERIAS '15, pp. 41:1–41:1, CERIAS - Purdue University.

47. Yadigar Imamverdiyev, "A hypergame model for information security," *International Journal of Information Security Science*, vol. 3, no. 1, pp. 148–155, Sep 2013.

48. Danda B. Rawat, Joel Rodriques, and Stojmenovic Ivan, *Cyber-physical systems: from theory to practice*, CRC Press/Taylor & Francis Group, 2016.

49. Wenyuan Xu, Timothy Wood, Wade Trappe, and Yanyong Zhang, "Channel surfing and spatial retreats:defenses against wireless denial of service," *Proceedings of the 2004 ACM workshop on Wireless security - WiSe '04*, 2004.

50. Carlo Kopp, "Shannon, hypergames and information warfare," in *3rd Australian Information Warfare & Security Conference (IWAR 02)*, IWAR 02.

51. Keith W. Hipel and Niall M. Fraser, "Conflict analysis, models and resolution," 1984.

52. IBM, *IBM ILOG CPLEX Users' Guide*, 7. IBM Corp, 12 edition, 2016.

53. Mikhael Shor, "Deadlock," `http://www.gametheory.net/dictionary/Games/Deadlock.html`, Aug 2005.

54. Sudeeptha Rudrapattana, "Cyber-security analysis in smart grid scada systems: A game theoretic approach," M.S. thesis, Texas Tech University, 2013.

55. Seth T Hamman, *Improving the Cybersecurity OF Cyber-Physical Systems Through Behavioral Game Theory and Model Checking in Practive and in Education*, Ph.D. thesis, Air Force Institute of Technology, 2016.

56. Michael D Wittman, "solving the blotto game: A computational approach," `http://web.mit.edu/wittman/www/Wittman_BlottoPaper.pdf`, Apr 2011.

57. Nicholas S. Kovach, *A Temporal Framework for Hypergame Analysis of Cyber Physical Systems in Contested Environments*, Ph.D. thesis, Air Force Institute of Technology, 2016.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23-03-2017	Master's Thesis	Sept 2015 — Mar 2017

4. TITLE AND SUBTITLE

Hypergame Analysis of Cyber Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Umodu, Kebin, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-17-M-075

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A:
Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

As technology is increasingly integrated into modern day systems, cyber defense becomes more important. Hypergame theory is a method that can model cyber attacks where players have different perceptions about the game being played. Capturing perceptions improves the accuracy of a model, and using this additional information a network administrator can formulate improved plans to better protect a cyber-physical system (CPS). AFIT's Hypergame Analysis Tool(HAT) software models these scenarios in a normal form hypergame. This investigation analyzes how HAT can be used to find more information for a network administrator defending a CPS. This scenario is modeled using a new cyber defense modeling framework (CDMF). HAT is upgraded with a mixed integer programming method (MIP) to find multiple Nash equilibriums(NE). MIP in HAT is used on cyber models to experiment with changing NEs and row players. The results show new ways of analyzing a hypergame and it's results in order to support the network administrator in making better defense plans against an unpredictable attacker.

15. SUBJECT TERMS

HyperGame Theory, Cyber Defense, Nash Equilibrium

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. G. B. Lamont, AFIT/ENG
U	U	U	UU	152	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4718; gary.lamont@afit.edu